UNIVERSIDAD AUTONOMA DE MADRID

ESCUELA POLITECNICA SUPERIOR

PROYECTO FIN DE CARRERA

Registro en tiempo real de señalización biológica utilizando una tarjeta de adquisición de datos USB

David Mancebo Calle
Diciembre 2013

Registro en tiempo real de señalización biológica utilizando una tarjeta de adquisición de datos USB

AUTOR: David Mancebo Calle
TUTOR: Pablo Varona Martínez

Grupo de Neurocomputación Biológica (GNB)
Dpto. de Ingeniería Informática
Escuela Politécnica Superior
Universidad Autónoma de Madrid
Diciembre de 2013

Resumen:

El objetivo de este proyecto es el diseño, la construcción y validación de un driver USB de adquisición/emisión de señales en tiempo real para la familia de tarjetas USBDUX. Este driver se puede utilizar en el registro/estimulación de sistemas biológicos y se prueba en una neurona electrónica. El sistema implementado en tiempo real *soft* consta de una DAQ (placa de adquisición de datos) USBDUX, en concreto la USBDUX-sigma, un PC con Linux Ubuntu 10.04 con patch RTAI y librerías COMEDI y las correspondientes conexiones necesarias para comunicar los dispositivos. También se emplea la USBDUX-fast para el registro de las señales de las pruebas de validación realizadas.

Para determinar las posibilidades y los límites del sistema en tiempo real desarrollado, se han realizado pruebas en varias configuraciones que permiten los componentes del sistema. Por un lado se implementa el sistema sobre las funciones y estructuras temporales que ofrece un GPOS, sistema operativo de propósito general. A su vez se realizan las mismas simulaciones sobre un sistema operativo en tiempo real (Ubuntu con patch RTAI) con garantías temporales reales necesarias para obtener un nivel de precisión adecuado. El modo de temporización del sistema puede ser implementado de dos maneras que utilizan modos de transferencia USB diferentes. Se validó la implementación del sistema en modo síncrono (temporización impuesta por el PC y modo USB bulk) y en modo asíncrono (temporización impuesta por la DAQ, y modo USB isócrono). El uso de sistemas dedicados, en general, ofrece mejores resultados. Toda la experimentación es realizada sobre un sistema uniprocesador y sobre un PC con mayores prestaciones y varios procesadores.

Se concluye que el sistema es adecuado para la adquisición/emisión con precisión temporal en el rango de los milisegundos pero no alcanza la precisión necesaria y los requerimientos de tiempo real estrictos para implementar ciclos cerrados en este rango de especificación temporal.

Palabras Clave:

Tiempo real *soft*, USB, Comedi, RTAI, USBDUX, Adquisición/emisión de alta precisión, ciclos cerrados de estimulación dependiente de actividad, sistemas de adquisición de bajo coste.

Abstract

The goal of the project is to design, implement and validate a driver for real time signal acquisition/delivery suitable for the USBDUX DAQ board family. This driver can be used to record/stimulate biological systems and is tested in an electronic neuron. The system is implemented in soft real time and consists of a USBDUX DAQ (data acquisition device), the USBDUX-sigma, a PC with Ubuntu 10.04 Linux with the RTAI patch and Comedi libraries, and dedicated wires to allow the connectivity between the components. The USBDUX-fast DAQ is also used for signal acquisition in the validation experiments.

Different tests have been performed in several configurations to assess the possibilities and limitations of the implemented real time system. On the one hand, the system has been implemented using the time control offered by a GPOS (general purpose operating system). On the other hand, the same tests have been run in a real time operating system that guaranties the time constraints to achieve the required accuracy. The timing system can be implemented in two different ways, each of them uses a different USB transmission mode. The system was tested in the synchronous mode (the PC handles the time slot of the whole system working in a USB bulk mode) and in the asynchronous mode (the DAQ handles the system time working in a USB isochronous mode). In general, embedded systems produce better results. A comparative test was made using a uniprocessor PC and a high-performance quad-core PC.

The system reaches the millisecond accuracy range in the validation experiments for stimulation and recording. However, the system is unable to achieve the same temporal accuracy in closed-loop experiments.

Key words

Soft real time, USB, Comedi, RTAI, USBDUX, high accuracy acquisition/emission, goal driven closed-loop, low-cost acquisition systems.

Agradecimientos

A mi familia.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1 Introducción

1.1 Motivación

La captación de señales biológicas en ciclos cerrados estímulo-respuesta de gran duración proporciona una visión más certera del comportamiento del fenómeno fisiológico que se esté produciendo. Debido a este supuesto en el laboratorio del GNB se lleva a cabo todo tipo de experimentos en los que se realizan ciclos de pruebas de estimulación y captación de las señales de la reacción en respuesta a un estímulo (Chamorro et al., 2012a; Muniz et al., 2008; Chamorro et al., 2009; Muniz et al., 2009).

Para la realización de estas lecturas de señales biológicas es necesaria una tarjeta de adquisición de datos (DAQ) que permita adquirir un grupo de señales analógicas, procesarlas y enviarlas a un PC para ser utilizadas en cualquier aplicación software. Cada señal biológica tiene una técnica diferente de captación. Es necesario el empleo de sensores específicos por lo que cada señal tendrá características independientes. El registro de estas señales debe ser controlado en todo momento. Los sensores tienen características muy diversas tanto en lo que se refiere a la captación de la señal como al tipo de conector. En general, las señales emitidas no se encuentran adaptadas a los rangos de señal capaces de leer la DAQ.

En la actualidad existen DAQ muy potentes y con grandes prestaciones, estas placas resultan muy costosas, por su tamaño son poco manejables y no están preparadas para realizar pruebas de campo. Por estas razones para este proyecto se utilizará una placa de adquisición de datos de bajo coste que se conecta a través de USB con suficientes prestaciones para poder manejar un buen rango de señales y que ofrece mayor dinamismo que las DAQ tradicionales.

La adquisición de estas señales se realizará por medio de una placa de la familia USBDUX, una placa de adquisición de datos totalmente compatible con COMEDI. COMEDI (http://www.comedi.org) es un proyecto de software libre con librerías de funciones para el desarrollo de drivers para el control y realización de mediciones en diferentes placas de adquisición o periféricos.

La captación de señales biológicas requiere de una gran precisión, al igual que los ciclos cerrados de estimulación-respuesta para la extracción de información o el control de sistemas biológicos (Muniz et al., 2009, 2008). El manejo de los tiempos de ejecución de las diferentes procesos es crítico; la imprecisión temporal o el jitter en las señales afectan los resultados de los experimentos, por esta razón es necesario emplear algún tipo de medio que dote al sistema operativo del manejo de los tiempos precisos del sistema, es decir es necesario el uso de un sistema de tiempo real.

RTAI (https://www.rtai.org) es una extensión al kernel de Linux que permitirá que el driver se ejecute en tiempo real, y en particular permitirá adquirir y enviar estimulación dependiente de actividad mediante la tarjeta con restricciones temporales estrictas. RTAI y las librerías COMEDI se emplearán como soporte necesario para crear un programa de adquisición de datos en tiempo real sobre diferentes señales biológicas.

1.2 Objetivos

El proyecto propuesto tiene como objetivo el diseño, la construcción y validación de un driver de adquisición de señales biológicas en tiempo real sobre un sistema operativo basado en LINUX (Ubuntu) y a través de USB. Para la adquisición y emisión de datos se utilizara una DAQ (placa de adquisición de datos) del tipo USBDUX totalmente compatible e integrada en LINUX gracias a las librerías COMEDI con funciones para el desarrollo de drivers para el control y realización de mediciones en diferentes placas de adquisición o periféricos. Este driver funciona en un sistema operativo de tiempo real bajo RTAI.

En estas condiciones de control y precisión temporal que ofrece RTAI, el driver para las tarjetas USBDUX es probado y utilizado para la captación de señales biológicas como la actividad del sistema nervioso simulada con una neurona artificial desarrollada en el Grupo de Neurocomputación Biológica (GNB) o un generador de funciones. A su vez la estimulación de alta precisión temporal es implementada y testada por medio de diferentes mecanismos de validación. Una vez demostrada la viabilidad de la emisión/adquisición de alta precisión, se realiza un sistema en tiempo real soft que ejecutará ciclos cerrados de estimulación de alta precisión dependiente de la actividad y gobernados por objetivo.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- Estado del arte: proporciona una visión global y esquemática de la temática en la que engloba el proyecto: DAQ's, sistemas en tiempo real y sus sistemas operativos, el protocolo USB, estimulación/adquisición de alta precisión y ciclos cerrados.

- Diseño: Se analizan las posibilidades que ofrecen las herramientas elegidas para el desarrollo del sistema. Se realiza un análisis de las características y parámetros que determinan la precisión y los ciclos cerrados. Para analizar en detalle se divide el capítulo en análisis del hardware del sistema: DAQ's, conexiones, características físicas de señal…; y en software: `posibilidades de manipulación de las DAQ's y funciones de Comedi, propiedades y utilidades de RTAI, patch real time elegido, modos de configuración del sistema, etc.

- Desarrollo: Se exponen las decisiones y soluciones empleadas para la elaboración del sistema. Se presenta en formato idéntico al diseño.

- Integración, pruebas y resultados: se comprueba empíricamente por medio de la implementación de un sistema de tiempo real de adquisición/emisión de precisión de milisegundos a través de USB. Una vez demostrada la viabilidad del sistema y caracterizadas sus limitaciones se realiza la implementación de ciclos cerrados de estimulación dependientes de la actividad y gobernados por objetivo.

- Conclusiones y trabajo futuro: Se apuntan y resumen las ideas futuras a partir de las posibilidades y limitaciones del sistema.

2 Estado del arte

2.1 Estimulación de alta precisión temporal

Los sistemas en tiempo real tienen un gran potencial para el desarrollo de aplicaciones con restricciones temporales estrictas necesarias para lograr una autentica estimulación controlada en sistemas complejos. En la industria actual encontramos gran cantidad de ejemplos en los que se desarrollan sistemas que realizan acciones que necesitan de una precisión temporal estricta, sistemas de control de satélites, robots industriales, maquinaria médica o sistemas de seguridad son ejemplos de ello. La falta de un control temporal preciso puede conllevar situaciones desastrosas o funcionalidad limitada. En el laboratorio del GNB se trabaja con varias situaciones en las que la estimulación de alta precisión temporal, en un rango de milisegundos, es necesaria para la implementación de ciclos cerrados donde el estímulo es dependiente de la respuesta en sistemas biológicos.

La estimulación de alta precisión temporal se realiza por medio de un control estricto de las situaciones que pueden acontecer en el sistema y la implementación de mecanismo de ejecución de tareas (fundamentalmente relativa al envío de señales) en función de su prioridad. Esto se logra por medio de RTOS (*real time operiting systems*, sistemas operativos de tiempo real). Gracias al uso de RTOS y conocidos los peores tiempos de ejecución de las tareas, se puede lograr un nivel de precisión en los tiempo de latencia de la emisión y adquisición de señales. El RTOS es capaz de determinar y asegurar la ejecución de tareas en intervalos temporales conocidos y estudiados controlando la incertidumbre temporal o *jitter*.

La precisión temporal está determinada tanto por la duración de los estímulos como por el momento de actuación deseado o la periodicidad de los mismos. Bajo el control del sincronismo que ofrecen los RTOS y por medio de DAQs, *data adquisition boards* o tarjetas de adquisición de datos, se pueden implementar sistemas de emisión/recepción de señales precisas donde la precisión está presente tanto en la duración como en el momento y el periodo de actuación que se requiera en la estimulación. Este tipo de sistemas de estimulación o control de alta precisión temporal pueden ser activados de manera periódica, por medio de una señal de sincronismo, o de manera esporádica por medio de interrupción hardware. El nivel de precisión también estará sujeto por tanto al tiempo que se necesita desde que se recibe la orden de estimular y el momento que se produce el estímulo. Estos mecanismos de estimulación o control pueden aplicarse de manera similar a la adquisición síncrona de muestras o en sistemas de envío dependientes de la adquisición. En definitiva, la estimulación de alta precisión temporal requiere de un sistema en tiempo real que gobierne y controle su actuación. Una vez lograda esta precisión temporal también se analiza la posibilidad de elaborar ciclos cerrados de estimulación dependientes de la actividad y condicionados por objetivo.

2.2 Ciclos cerrados de estimulación dependiente de la actividad.

Uno de los ejemplos más representativos de tecnología de ciclo cerrado, o estimulación dependiente de actividad, en neurociencia es el concepto de *dynamic-clamp* o pinzamiento dinámico en los que, mediante la inyección de corriente en función del potencial registrado, se introducen conductancias iónicas o sinápticas artificiales en neuronas biológicas (Robinson and Kawai, 1993; Sharp et al., 1993; Destexhe and Bal, 2009). Este ciclo se ha de construir con una precisión temporal alta para que las conductancias resultantes sean realistas. En el GNB se ha ampliado el concepto clásico de dynamic clamp, para desarrollar nuevos protocolos que se aplican a diferentes contextos de investigación pero todos bajo ese principio de ciclo cerrado de estimulación/respuesta de alta precisión temporal dirigida por un objetivo a conseguir (Chamorro et al., 2012a).

Estos protocolos de estimulación de alta precisión revelan aspectos de la dinámica de los sistemas biológicos que aparecen ocultos en los protocolos tradicionales de estímulo respuesta en ciclo abierto. Existen procesos imposibles de observar por medio de sistemas tradicionales de estimulación-respuesta en ciclo abierto que no contemplan los aspectos de procesamiento dependiente de la historia previa en la dinámica de las neuronas y redes neuronales. Estos sistemas son no lineales y adaptativos, trabajan en regímenes transitorios y solo se puede observar los estados de manera parcial por medio de una serie de variables que caracterizan el momento concreto en el que se encuentra el sistema (Chamorro et al., 2012a). Los protocolos de estimulación de alta precisión en ciclos cerrados dependientes de la actividad registrada *online* son utilizados para revelar ritmos o dinámica de un amplio rango de procesos, y para lograr un control de estados patológicos o naturales. De la misma forma estos protocolos se pueden utilizar para favorecer el aprendizaje y también para automatizar experimentos (Fernandez-Vargas et al., 2013).

En particular, en el GNB se ha mostrado la eficiencia de los ciclos cerrados en tres tipos de experimentos diferentes, todos empleando protocolos adaptativos implementados en un ciclo que monitoriza y caracteriza la dinámica o el comportamiento del sistema en una escala de hasta unos pocos milisegundos. Con esta información se obtiene una representación y un control más efectivos del sistema. Gracias a este conocimiento del estado instantáneo del sistema, de su evolución, y al uso de sistemas de RT se puede controlar o precisar en función del objetivo el nivel de intensidad, duración o periodicidad de un estímulo en relación al objetivo dado al ciclo cerrado. Los experimentos se han realizado en los siguientes ámbitos (Chamorro et al., 2012a):

- Ciclos cerrados para la microinyección de neurotransmisores o neuromoduladores en preparaciones *in vitro*.

- Ciclos cerrados para la estimulación gobernada por eventos detectados por video en neuroetología.

- Ciclos cerrados de estimulación mecánica para el estudio de la transformación senso-motora.

El sistema entero de experimentación basado en ciclo cerrado de estimulación de alta precisión gobernado por objetivo queda ilustrado en el esquemático de la Figura 1.

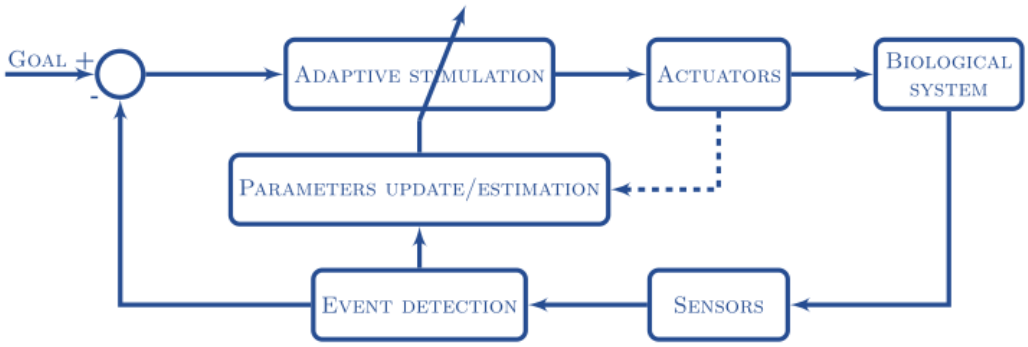


Figura 1: representación esquemática de un ciclo cerrado de estimulación dependiente de la actividad gobernado por objetivo. Especificado un objetivo, se adquiere la señal a través de unos sensores y una DAQ, se detectan los eventos relevantes, se actualizan los parámetros de la representación interna de la interacción, se ajusta la estimulación o el explorador de estímulos y se envían las señales de control a los dispositivos actuadores(Chamorro et al., 2012b).

En los tres ejemplos anteriores se implementa ciclos cerrados de este tipo. En los ciclos cerrados de estimulación gobernados por objetivo, la actividad biológica o en general el estado de un sistema a estudiar son monitorizados a través de un conjunto de sensores que caracterizan distintos tipos de fenómenos. Las señales de control o estimulación del sistema son emitidas a través de actuadores específicos (muchos de los cuales necesitan recibir señales de control cada pocos milisegundos). Fijado un objetivo de análisis la detección de un determinado evento, se utiliza para lanzar la estimación que conduce a un estado concreto del proceso biológico estudiado (Muniz et al., 2008; Chamorro et al., 2012a).

2.3 Tarjetas de adquisición de datos

La adquisición de datos es el proceso de medir o cuantificar un fenómeno eléctrico o físico como voltaje, corriente, temperatura, presión; cualquier fenómeno cuyo cambio pueda ser detectado por algún sensor. Estos sensores generan señales analógicas en la que se refleja las alteraciones que ocurren en el fenómeno de estudio. La Guía Completa para Construir un Sistema de Medidas de National Instruments, (http://www.ni.com/data-acquisition/esa/) proporciona una visión general sobre los sistemas de medición mediante tarjetas e adquisición de datos. Un sistema de adquisición se compone típicamente de una serie de sensores o transductores, encargados de convertir fenómenos físicos en señales eléctricas que se puedan medir, la tarjeta DAQ (data acquisition) o tarjeta de adquisición de datos, encargada de la traducción de la señal eléctrica analógica a una muestra digital, y un PC que procesa la información digital. El hecho de incorporar un PC en estos sistemas de adquisición hace que se aumente la potencia y sobre todo la flexibilidad del sistema de medida, ya que las posibilidades de procesamiento de información, la sincronización de tiempos y la interactuación con otros sistemas aumentan considerablemente teniendo en cuenta que los procesadores actuales trabajan a velocidades muy altas.

La DAQ o tarjeta de adquisición es la interfaz que relaciona el mundo analógico con el PC. Su función principal es la de digitalizar las muestras analógicas adquiridas. En la actualidad se las dota de otras funcionalidades como la capacidad de emisión de señales analógicas para automatizar/controlar sistemas de medida o procesos. Con estos canales analógicos de entrada y salida se pueden implementar sistemas de control y sistemas de experimentación basados en ciclos cerrados de estimulación conducida por objetivo.

El mecanismo de adquisición de datos está dividido típicamente en tres componentes claves (ver Figura 2):

- Circuito de acondicionamiento de Señal
- Conversor Analógico Digital (ADC)
- Bus de PC

Figura 2 : Ejemplo de los componentes básicos de un sistema de adquisición de datos obtenida de la página web del fabricante de DAQ's, National Instrument (www.ni.com/data-acquisition/what-is/esa/).

En la actualidad existe una gran variedad de tarjetas DAQ en el mercado con prestaciones muy variadas, tanto en velocidades de transmisión de datos entre la tarjeta y el PC, como en lo referente a las tasas de adquisición y al tipo y rango de señales que puede manejar. Incluso se las dota de nuevas funcionalidades como la emisión de señal analógica o la posibilidad de trabajar con señales digitales. Estas tarjetas pueden ser utilizadas para el control o estimulación de dispositivos de muy diverso tipo como motores de paso, robots y maquinaria industrial.

En la guía Completa para Construir un Sistema de Medidas, de National Instrument se pueden encontrar unas recomendaciones en forma de preguntas para la elección de la DAQ necesaria para nuestro cometido. Estas preguntas se resumen en qué tipo de señales se requiere medir o generar, la necesidad de implementar un acondicionamiento de señal, los requisitos de velocidad de adquisición o generación de impulsos del sistema, el umbral de detección, y por último la tolerancia al error.

Empresas como Measurament Computing o National Instruments ofrecen sistemas completos de adquisición de datos y monitorización con muy buenas prestaciones, tanto a

nivel software como hardware, sistemas que poseen licencias de uso, por lo que representan soluciones costosas. Por otro lado, la familia de tarjetas DAQ de bajo coste USBDUX, (http://www.linux-usb-daq.co.uk) está diseñada con filosofía open-source y para trabajar sobre Comedi (www.comedi.org), un proyecto de software libre integrado en Linux. Comedi, control and measurement device interface, es un conjunto librerías, herramientas y drivers que posibilita el manejo de una gran variedad de tarjetas DAQ, de National instrument, de Measurament Computing, de la familia USBDUX y otros fabricantes hardware (www.comedi.org/hardware.html) en sistemas operativos Linux.

El protocolo de comunicación entre la DAQ USBDUX y el PC se realiza a través de USB por lo que son una buena herramienta para el desarrollo de sistemas de adquisición/control con gran flexibilidad y portabilidad por su reducido tamaño. La gama USBDUX es de bajo coste, conectable al PC por medio de USB, y ofrece unas buenas prestaciones en cuanto a rangos de señal, tasas de muestreo y emisión de señal, todo bajo el control de software libre con driver de Comedi y entornos Linux. Al estar integrado en Comedi se posibilita el uso de las tarjetas en un entorno de tiempo real, en concreto a través la extensión de kernel RTAI (www.rtai.org) como se detallará más adelante (véase 2.5.8 Real Time Aplication Interface. RTAI).

2.4 Protocolos de comunicación con DAQ's

Existen varios protocolos de comunicación entre el PC y la DAQ. La elección del protocolo correcto depende de varios factores a valorar como ancho de banda necesario para el volcado de información de la DAQ al PC, el nivel sincronización de los componentes, los requerimientos necesarios para la entrada/salida, la portabilidad del sistema y distancias entre los diferentes sistemas a monitorizar o estimular. La Tabla 1 ilustra como ejemplo las características de los protocolos de comunicación en las tarjetas de National Instruments.

Bus	Waveform[1] Streaming	Single-Point I/O	Multidevice	Portability	Distributed Measurements	Example
PCI	132 MB/s (shared)	Best	Better	Good	Good	M Series
PCI Express	250 MB/s (per lane)	Best	Better	Good	Good	X Series
PXI	132 MB/s (shared)	Best	Best	Better	Better	M Series
PXI Express	250 MB/s (per lane)	Best	Best	Better	Better	X Series
USB	60 MB/s	Better	Good	Best	Better	NI CompactDAQ
Ethernet	125 MB/s (shared)	Good	Good	Best	Best	NI CompactDAQ
Wireless	6.75 MB/s (per 802.11g channel)	Good	Good	Best	Best	Wi-Fi DAQ

[1]Maximum theoretical data streaming rates are based on the following bus specifications: PCI, PCI Express 1.0, PXI, PXI Express 1.0, USB 2.0, Gigabit Ethernet, and Wi-Fi 802.11g

Tabla 1: Características de los protocolos de comunicación entre DAQ y PC en las tarjetas de National Instruments.

En este proyecto se utilizarán la tarjetas DAQ USBDUX, las cuales emplean el protocolo de comunicación USB para realizar la transferencia de datos con el PC, para la implementación de sistema de monitorización y estimulación.

2.4.1 USB

El protocolo USB, *universal serial bus*, está diseñado para implementar una comunicación con todo tipo de periféricos sin los límites que tenían los antiguos interfaces o protocolos de comunicación clásicos (www.usb.org). Las principales ventajas que ofrece USB son la facilidad de uso, la posibilidad de gestión de múltiples periféricos, la configuración automática, y la facilidad de conexión. El USB es considerado *hot plugged*, por lo que puede ser conectado y desconectado sin producirse fallos en el hardware del periférico, no necesita configuración por parte del usuario, permite la utilización de recursos por otros periféricos, y puede proporcionar alimentación eléctrica. Las velocidades teóricas que ofrece el protocolo depende de en qué versión del mismo nos encontremos: para modos high-speed, la velocidad del bus es de 480 Mbytes/s que corresponde con la velocidad de transferencia del bus en la versión 2.0. También en 2.0 se puede manejar full- speed con 12 Mbytes/s o low speed con 1.5 Mbyes/s. El protocolo ofrece un mecanismo de seguridad para asegurar a transferencia de datos correcta. Por otro lado es una tecnología muy extendida que se encuentra en todos los PC actuales.

Desde de punto de vista de desarrollo, el protocolo USB tiene soporte en una gran cantidad de sistemas operativos. Cualquier sistema operativo que soporte USB tiene ser capaz de realizar la detección de conexión y desconexión del periférico, la comunicación con el periférico registrado, el establecimiento de un mecanismo para la transferencia de datos y la comunicación con el periférico a través de los controladores software. A su vez nos ofrece una gran versatilidad en el modo de transferencia de los datos desde el periférico al host PC. Existen cuatro modos de transferencia en el protocolo USB y sus características quedan resumidas en la tabla 2 (Axelson, 2005):

- Modo control:

Se utiliza para recibir las peticiones de comunicación con las especificaciones USB necesarias para que el host conozca las características del dispositivo. Realiza las funciones de identificación y configuración.

- Modo Bulk:

Este modo es útil cuando se quieren transmitir datos en situaciones en las que el tiempo no es crítico. En este modo se puede enviar grandes cantidades de datos sin colapsar el canal de transferencia debido a que este modo puede retrasar su emisión respecto a otros modos de transferencia hasta que haya un espacio de tiempo adecuado para el envío. A su vez posee detector y corrector de errores.

- Modo interrupciones:

Este modo es utilizado en aplicaciones en las que los datos tienen que ser enviados o recibidos en un plazo máximo. Se utiliza en periféricos que necesitan atención periódica para su funcionamiento. Este modo de transferencia se ejecuta a cualquier velocidad.

- Modo isócrono:

La transferencia isócrona es fluida y con latencias fijas, este modo se utiliza en aplicaciones en las que los datos deben llega a un ritmo constante o en un momento concreto, y que puede tolerar fallos ocasionales.

Las principales características de los cuatro modos se exponen en la siguiente tabla:

Transfer Type	Control	Bulk	Interrupt	Isochronous
Typical Use	Identification and configuration	Printer, scanner, drive	Mouse, keyboard	Streaming audio, video
Required?	yes	no	no	no
Low speed allowed?	yes	no	yes	no
Data bytes/millisecond per transfer, maximum possible per pipe (high speed).*	15,872 (thirty-one 64-byte transactions/ microframe)	53,248 (thirteen 512-byte transactions/ microframe)	24,576 (three 1024-byte transactions/ microframe)	24,576 (three 1024-byte transactions/ microframe)
Data bytes/millisecond per transfer, maximum possible per pipe (full speed).*	832 (thirteen 64-byte transactions/ frame)	1216 (nineteen 64-byte transactions/ frame)	64 (one 64-byte transaction/ frame)	1023 (one 1023-byte transaction/ frame)
Data bytes/millisecond per transfer, maximum possible per pipe (low speed).*	24 (three 8-byte transactions)	not allowed	0.8 (8 bytes per 10 milliseconds)	not allowed
Direction of data flow	IN and OUT	IN or OUT	IN or OUT (USB 1.0 sup- ports IN only)	IN or OUT
Reserved bandwidth for all transfers of the type (percent)	10 at low/full speed, 20 at high speed (minimum)	none	90 at low/full speed, 80 at high speed (isochronous & interrupt combined, maximum)	
Error correction?	yes	yes	yes	no
Message or Stream data?	message	stream	stream	stream
Guaranteed delivery rate?	no	no	no	yes
Guaranteed latency (maximum time between transfers)?	no	no	yes	yes
*Assumes transfers use maximum packet size.				

Tabla 2: Características de modos de transferencia de datos sobre el protocolo USB(Axelson, 2005).

2.5 Sistemas en tiempo real

En la actualidad existen una gran cantidad de procesos y maquinaria que requieren de una precisión temporal estricta para poder realizar su cometido correctamente: robots que controlan procesos químicos o que monitorizan procesos, satélites que utilizan señales calibración, sistemas de seguridad en vehículos como el airbag necesitan interactuar con el entorno en un momento en concreto para lograr el objetivo para el que están diseñados, etc. Por lo tanto, en los sistemas de tiempo real el correcto funcionamiento del sistema no depende únicamente de la salida del sistema, sino también de en qué instante de tiempo ocurre. Es importante comprender que cuando hablamos de sistemas en tiempo real no hablamos de sistemas que procesan la información muy rápido, sino de que el sistema está

diseñado para poder realizar su función en el momento que se requiera, sistemas precisos y predecibles en los tiempos de ejecución de sus tareas (Hansson et al., 2010) .

Es complicado hablar del estado del arte de los sistemas en tiempo real ya que son implementados de maneras muy diferentes dependiendo de las características de la precisión temporal o del nivel de sofisticación en prestaciones que se requiera para realizar tarea para la que este destinado el sistema en tiempo real. En general, podemos hablar de un sistema microprocesador que maneja conjuntos cerrados de instrucciones y que maneja una rutina de interrupciones (Hansson et al., 2010) .

Dependiendo de las especificaciones particulares del sistema, los sistemas en tiempo real se pueden considerar hard o soft real time, la diferenciación nace de cuán estrictas son las especificaciones del sistema. En ambos casos se obtiene un sincronismo con el entorno sobre el que actúa (Korver, 2003).

2.5.1 Hard real time

En los sistemas hard real time tienen una respuesta determinística ante cualquier evento. Los sistemas hard real time deben tener total sincronismo con el entorno y analizar todas las situaciones críticas. La seguridad y la temporización del sistema deben estar aseguradas. Estos sistemas son implementados en aplicaciones industriales, en automoción, para instrumentación médica, etc. Se puede considerar que son sistemas en los que la incertidumbre o el *jitter* en los tiempos de actuación de las tareas que forman el sistema conllevan a un colapso total del sistema. Otra forma de sincronismo con el entorno menos estricto, la implementan los sistemas soft real time que se detallan en la siguiente apartado.

Los sistemas de airbag cuentan con un requerimiento temporal estricto, en el que una actuación excesivamente rápida puede agravar los daños que ocurran sobre los usuarios del vehículo durante el accidente y una reacción tardía conlleva causas fatídicas. Este tipo de sistemas se conocen como hard real time o sistemas estrictos de tiempo y la perdida de las constantes temporales no es una opción ya que controlan actividades o procesos críticos, cierto que en este caso del airbag su implementación se realiza con hardware dedicado pero existen muchos ejemplos de sistemas hard real time elaborados en software por ejemplo los experimentos de ciclo cerrado llevados a cabo en el GNB. Imaginemos ahora un sistema de video, el jitter que pueda sufrir la transmisión de datos para su proyección no representa una situación crítica, el cumplimiento de las restricciones temporales no es una condición necesaria para el correcto funcionamiento del sistema aunque sí que afecta en la calidad del mismo. Este tipo de sistemas son conocidos como soft real time.

2.5.2 Soft real time

En los sistemas soft real time la respuesta del sistema no es totalmente determinista ante un evento sino que se habla de tiempos de respuesta o latencia en media. En este tipo de sistemas la perdida de datos se puede tolerar y no conllevan a situaciones desastrosas.

El protocolo USB no está diseñado para aplicaciones hard real time sino para sistemas operativos de propósito general, por esta razón solo será posible implementar un sistema

soft real time a pesar de que existan modos que ofrezcan latencias de transmisión fijas.(Korver, 2003) .

Esta diferenciación de sistemas en tiempo real hard vs soft no es la única clasificación. Podemos hablar de sistemas en tiempo real en los que el evento dispara la acción que realizará el sistema en tiempo real, *event-triggered* y sistemas que funcionaran bajo el control de un mecanismo temporal que controla las constantes temporales del sistema: los eventos ocurren en momentos de tiempo conocidos, *timed-triggered*. Dependiendo del nivel de especificación y el tipo de soporte del sistema podemos hablar de sistemas embebidos en tiempo real, mini ordenadores especializados en la cometida de una serie de funciones determinadas o no embebidos como los PC que tienen un propósito general y sus tareas pueden ser reprogramadas.

2.5.3 Propiedades de los sistemas en tiempo real

Hanson y colaboradores definen los sistemas de tiempo real de acuerdo con las siguientes características (Hansson et al., 2010):

- Ejecución predecible: para un buen funcionamiento del sistema en tiempo real es necesario asegurar el control de los tiempos de ejecución de las diferentes tareas y las consecuencias que conlleve la ejecución de esas tareas, como por ejemplo la utilización de recursos. Se conoce toda la estructura temporal del sistema y se elabora una planificación de las tareas a ejecutar basada en que todas las tareas cumplan con sus requisitos temporales.

- Multitarea: en un sistema real, varias tareas compiten por la utilización de recursos del sistema, como pilas o posiciones de memoria. Las tareas van a ser ejecutadas en paralelo. No se trata de un paralelismo real como el que ocurre en los procesadores de varios cores sino de intercambiar muchas veces y muy rápido de tarea que se ejecuta en cada momento.

- Priorización: dentro del sistema hay tareas que tiene mayor relevancia para el buen funcionamiento del sistema por lo que tienen mayor prioridad de ejecución asignada.

- Tipo de fuente que lanza la ejecución de la tarea: podemos hablar de dos principios fundamentales de cómo controlar sistemas en tiempo real:
  - event-triggered: sistemas en los que eventos externos monitorizan y planifican la ejecución de las tareas.
  - timed-triggered: sistemas temporizados en que las tareas se van ejecutando en momentos de tiempos concretos conocidos, en estos sistemas las tareas se ejecutan de acuerdo a una planificación antes de la ejecución por lo que los problemas de sincronización en el uso de recursos y comunicación entre procesos se solventan en tiempo de diseño.

También las tareas tienen asociados diferentes estados, *ready, suspend* o *wait* por ejemplo, mientras trabaja el sistema en tiempo real. En los RTOS es importante la fiabilidad del

sistema por lo que están diseñados para soportar picos de carga de información y no colapsar las cola de recepción de datos o registros del sistema.

2.5.4 Parámetros de caracterización en de sistemas en tiempo real:

En el artículo Adequacy of the Universal Serial Bus for real-time systems se analiza la posibilidad de utilización del protocolo USB para la implementación de sistemas en tiempo real y se analiza diferentes parámetros para caracterizar los límites, la eficacia y fiabilidad del protocolo en una determinada configuración (Korver, 2003). Existe una gran cantidad de parámetros de caracterización, en este apartado nos centraremos en los más relevantes para este proyecto:

Tiempos de latencia

El tiempo de latencia es el tiempo que se emplea en realizar una tarea específica, es por tanto el tiempo necesario para realizar el procesado de información y la posterior actuación una vez detectado un evento. Los tiempos de latencia pueden ser especificados de muchas maneras, una de las más utilizadas es el peor tiempo de ejecución WCET. El Worst Case Execution Time es el tiempo máximo de latencia calculado cuando se somete al sistema al mayor estrés posible, utilizando el máximo valor de datos y los escenarios de ejecución críticos. El WCET se calcula por medio del análisis del código fuente del programa, del número de instrucciones y recursos que utiliza, analizando el compilador y también la arquitectura del micro, aunque no es objetivo de este proyecto el cálculo teórico de esta situación sino que se analizará de manera práctica.

Jitter

Especifica la precisión de la latencia del sistema, es decir, mide las variaciones temporales que sufren los tiempos de ejecución o latencia de las tareas en el sistema. Este jitter o incertidumbre provoca que el sistema tenga un comportamiento menos preciso y por tanto predecible por lo que afecta a la integridad de los valores y características de un sistema en tiempo real. En sistemas *soft-real time* el jitter ha de ser tan pequeño como sea posible, sin embargo esto es algo crítico en sistemas estrictos de tiempo real.

Fiabilidad y robustez

Como en la mayoría de los sistemas la aparición de errores debe ser mínima. Una de las propiedades del sistema será la tolerancia al error que tenga. No todos los problemas que se presentan en el sistema tienen el mismo impacto en su funcionamiento. Errores que pueden aparecer son por ejemplo errores en la medida de los datos dependientes de la precisión del sistema, o también se pueden dar perdidas de muestras.

Otros parámetros que se utilizan para caracterizar el sistema en tiempo real son:

- Escalabilidad y flexibilidad. Un buen sistema en tiempo real debe poder adaptarse a procesos o tareas más sofisticados con el tiempo y a su vez poder ser integrado con nuevos componentes sin perder sus características temporales o funcionalidad.

- Estructura física: los canales de comunicación se pueden realizar sobre estructuras en anillo, punto a punto o en bus.

- Control de flujo o *flow control*. Es el control de la velocidad de la información que fluye entre el emisor y el receptor.

    - Explícito: el emisor impone sobre el receptor la velocidad del información sobre el canal y a su vez se ocupa de comprobar el que dato es recibido por el receptor
    - Implícito la transmisión de información se realiza a una velocidad constante y no se realiza una comprobación de la buena comunicación.

- Compatibilidad: la composición dela red de comunicación debe ser diseñada para un propósito y ofrecer unas características independientemente de cuantos componentes estén interconectados.

Como se indicaba anteriormente un sistema considerado hard real time no puede permitirse tiempos de latencia con grandes jitters y debe tener mecanismo de acción eficiente contra errores que en el caso de acontecer están medidos y controlados. La pérdida de un dato o de una temporización impuesta por el sistema provoca el colapso del mismo y puede desembocar en una situación fatídica. Imaginemos la perdida de una señal o retraso de la mima en un sistema de seguridad que activa el airbag en un coche tras la colisión. En sistemas en los que los errores no son controlables pero posibles y aceptables para el buen funcionamiento del sistema y no se pueda asegurar una latencia constante pero si una buena latencia media con un jitter medido son considerados sistemas soft real time.

2.5.5 Sistemas Operativos en tiempo real

Un RTOS (real-time operating system) o sistema operativo en tiempo real es un sistema operativo capaz de garantizar cierta funcionalidad cumpliéndose determinadas especificaciones temporales. Esto convierte a los RTOS la plataforma de desarrollo para sistemas en tiempo real.

Se considera que los RTOS se encuentran en un nivel intermedio entre el código de manejo del hardware del sistema y el nivel de aplicaciones del usuario o mundo software. El RTOS tiene accedo a las funciones de manejo de hardware que se encuentran en la HAL (Hardware Abstraction Law) como por ejemplo manipulación o manejo de registros o acceso al planificador de procesos para la asignación de tiempos de ejecución. A su vez el RTOS da servicio y soportes a las aplicaciones de tiempo real (Hansson et al., 2010).

Características de los RTOS:

- Manejo de las tareas o procesos. El sistema puede lanzar las tareas y asignarles prioridad para así poder controlar los tiempos y la eficacia de respuesta. Existen una gran variedad de políticas de planificación. La planificación se realiza en base a que se conocen los peores tiempos de ejecución y los tiempos de manejo de las interrupciones que puedan ocurrir.

- Manejo de los recursos del sistema. El sistema operativo tiene acceso y clasifica los diferentes componentes del sistema.
- Comunicación y sincronización: asegura buenos servicios de comunicación entre diferentes tareas y controla la sincronización para poder asegurar una cooperación entre las tareas o procesos

- Servicio de temporización del sistema, parte esencial del sistema en tiempo real. Otorgan el mecanismo para la emisión de señales de reloj.

2.5.6 GPOS VS RTOS

La principal diferencia entre los sistemas operativos es la necesidad del comportamiento determinista temporal que requieren los sistemas operativos en tiempo real. Esto hace que se tengan controlado en todo momento los tiempos que necesita el sistema operativo y las tareas que componen las aplicaciones software para realizar sus servicios.

El sistema operativo es el encargado de asignar tiempos de ejecución a las diferentes tareas. Esta función de planificación, en los sistemas operativos de propósito general, le otorga una prioridad más alta que cualquier tarea que se tenga que ejecutar en el sistema. Para su buen funcionamiento, es necesario planificar unos tiempos de ejecución de instrucciones asociadas a tareas que realiza el sistema operativo. El problema es que no es posible controlar con precisión temporal alta las tareas que realiza el sistema operativo ya que las tareas de control tienen asignadas mayor prioridad de ejecución.

Los sistemas operativos de propósito general no tienen un comportamiento determinístico temporal, lo que puede causar retrasos aleatorios a los tiempos de ejecución de las tareas que necesite realizar. Este *jitter* incontrolado provoca que el sistema sea poco fiable.

Otra diferencia es la resolución temporal con la actúa el sistema, en los sistema RTOS la resolución temporal es mayor que en los GPOS. El Clock Tick que representa la mínima unidad de tiempo del sistema suele ser de milisegundos en el caso de los RTOS mientras que en los GPOS la resolución es mayor. La planificación y sincronización de las tareas y recursos que maneja el sistema se realiza en función del sistema de reloj y su Clock Tick. En general, valores pequeños de resolución temporal ganan en sensibilidad y permiten operar con funciones periódicas con altas frecuencias de actuación (Hansson et al., 2010).

2.5.7 Alternativas de sistemas operativos en tiempo real

Debido a la importancia que han tomado los sistemas en tiempo real en los últimos años han aparecido una gran variedad de sistemas en tiempo real comerciales que ofrecen solución a una gran cantidad de necesidades. Estos sistemas operativos son implementados de manera independiente, creándose todo el sistema de soporte necesario, planificadores, colas de procesos, manejo de IRQ… o se han realizado a través de sistemas tradicionales de propósito general a los que se le añade funcionalidad específica para real time (Muñiz, 2005).

Las especificaciones requeridas imponen la necesidad de utilización de un sistema operativo en concreto. Si el sistema va a ser embebido con un procesador ARM el OSE (www.enea.com/solutions/rtos/ose/), operiting system embedded, es una solución muy extendida. Otra solución ofrecida desde un entorno Unix es el sistema operativo en tiempo real para embebidos lynxOS (www.lynuxworks.com/rtos/rtos.php).

Para soluciones de tiempo real que no necesiten ser embebidas, sino que funcionarán en PC comerciales, existe también una gran variedad de sistemas operativos desarrollados por diferentes compañías: xWork (www.windriver.com/), pSOS+ (www.dynatem.com/psosfs.html), QNX/Neutrino (www.qnx.com) o REAL/IX (www.modcomp.com/realtime/products/descriptions/8272.html). La mayoría de los productos de estas compañías son UNIX aunque no sean compatibles entre ellos.

En Linux, sistema operativo de propósito general basado en UNIX, que ofrece total libertad respecto al código, las licencias son públicas, tiene funcionalidad de multiprocesos y es un entorno de desarrollo muy extendido en la comunidad científica.
Windows también tiene posibilidades real time, como por ejemplo labVIEW-RT (http://www.ni.com/labview/realtime/esa/) ofrecido por National Instrument o Intime(http://www.tenasys.com/index.php/overview-ifw). El problema es que estas soluciones requieren de hardware y software específico y con un coste añadido. En entornos Linux se desarrollaron dos maneras para dotar al sistema de unas características y funcionalidad real time:

- Mejorando el nivel de concurrencia, caso de KURT(www.ittc.ku.edu/kurt/) que reduce el tiempo empleado en las partes donde no se puede dar concurrencia y así disminuyendo los tiempos de latencia.
- Añadiendo un patch de kernel real time sobre un kernel Linux, RTAI, Real Time Aplication Interface (www.rtai.org), RTLinux (www.rt.wiki.kernel.org) y Xenomai(www.xenomai.org).

En este caso se logra que las tareas del sistema operativo Linux se ejecuten como tareas de mínima prioridad del sistema. Las interrupciones hardware y software son manejadas a través del kernel RT añadido. Las principales características de las extensiones de tiempo real para kernel Linux se listan en la siguiente tabla:

	RTAI	RTLinux	Xenomai
¿Espacio usuario?	Soportado totalmente.	No soportado totalmente	Soportado totalmente.
Soporta varios planificadores	Uniprocesador. Multi-uniprocesador. Simetric Multi-uniprocesador	Uniprocesador.	Uniprocesador. Multi-uniprocesador. Simetric Multi-uniprocesador
Soporta varios HAL (hardware abstraction Layer)	RTAI HAL ADEOS	RTLinux	ADEOS
¿Soporte de DAQ's USBDUX?	Si, Comedi ofrece controlador	Si, Comedi ofrece controlador	No, RTDM basado Comedi (Analogy)

Tabla 3: principales diferencias entre RTAI, RTLinux y Xenomai.

RTAI es el sistema operativo en tiempo real utilizado en el grupo GNB. Las necesidades temporales que demanda el sistema de adquisición o emisión son abarcadas por las características temporales de RTAI. Además, en RTAI puede ser instalado Comedi, proyecto que ofrece un controlador y herramientas para el manejo de las tarjetas.

2.5.8 Real Time Aplication Interface. RTAI

Por una parte los GPOS ofrecen una gran cantidad de herramientas y recursos para realizar aplicaciones de usuario, pero no poseen el comportamiento determinístico temporal necesario para realizar tareas en las que se requiera un control temporal estricto, lo que sí ofrecen los RTOS. La extensión del kernel de propósito general de un sistema Linux posibilita la obtención de un sistema operativo de tiempo real. RTAI, Real Time Aplication Interface, se presenta como una solución intermedia en la que se puede realizar tareas con un control estricto temporal y a su vez continuar disfrutando de las herramientas de desarrollo que nos ofrece el mundo LINUX.

RTAI por medio de su Hardware Abstraction Layer consigue el control de todos los componentes hardware del PC situando la ejecución de sus instrucciones en una posición prioritaria respecto a las tareas o procesos que maneja Linux. La ejecución de las tareas por parte del procesador está totalmente controlada. En el laboratorio de GNB se utiliza este sistema operativo en tiempo real como soporte para la mayoría de la investigación (Muñiz, 2005) .

En RTAI también encontramos todo el soporte necesario en Comedi, controlador de las tarjetas DAQ USBDUX y un API de funciones para la manipulación de la tarjeta, Kcomedilib. RTAI permite el manejo de tareas soft y hard real time y es posible el desarrollo de aplicaciones en modo usuario. En el apartado de diseño se analizarán las características y cualidades de RTAI y los modos de adquisición en Comedi.

2.6 USB y tiempo real:

Ya se ha mencionado que el protocolo de comunicación USB posee cuatro modos de transferencia de datos. Como se describía en la Tabla 2 en el apartado USB, cada modo de transferencia tiene unas características determinadas (Axelson, 2005). No todos los modos de transferencia son adecuados para realizar sistemas de comunicación que sirvan para implementar sistemas en tiempo real. El modo de control sirve para los ajustes y la configuración de los periféricos. El modo de transferencia por bulk no puede asegurar una latencia o tiempo máximo entre transferencias entre el host PC y el periférico, por lo que es utilizado para el envío de grandes cantidades de datos a través del cable USB donde las tasas de transferencia no sean críticas. En el caso del modo interrupción existe una garantía de latencia por lo que puede resultar un modo apto para la elaboración de sistemas en tiempo real ya que el PC host y el periférico pueden manejar una atención periódica. El problema está en que posee un mecanismo corrector de errores. Este mecanismo produce un jitter incontrolable ya que la detección de error produce que se reenvie el paquete.

En el artículo Adequacy of the Universal Serial Bus for real-time systems (Korver, 2003)se extraen unas conclusiones acerca de la viabilidad del protocolo USB para el desarrollo de sistemas de tiempo real por medio del análisis práctico de una tarjeta FX2 (cypress) similar a la de las USBDUX pero se realiza la experimentación en una versión 1.1 de protocolo en un modo de transferencia isócrono.

El modo de transferencia isócrono tiene garantías de transferencia en tiempos fijados pero sin corrección de errores. Este es el único modo que no acepta reenvió de tramas erróneas. Este modo de transferencia es el utilizado para la reproducción de audio y video en tiempo real. En el libro USB Complete se dedica un apartado acerca de problemas de tiempo en la transferencia de datos sobre el protocolo USB(Axelson, 2005). En este apartado se expone diferentes componentes que pueden producir límites en la capacidad de envío y recepción de datos. Las capacidades del PC Host como tamaño de buffer de recepción o implementación de la tarea, son una característica importante y también las latencias que se ocasionan en el host PC. El manejo de las tareas por parte de los sistemas operativos de propósito general no puede asegurar con garantías la transferencia de datos con el periférico a una tasa concreta. El sistema operativo de propósito general es capaz de asignar tiempo de ejecución a los hilos en un sistema multitask en los que se da ejecución concurrente con una latencia en media de 1 milisegundo pero ocurriendo casos de latencias de hasta 100 milisegundos.

Un periférico USB y su software no tienen control sobre las tareas que el procesador está ejecutando en el PC host, por lo que tratar con esas latencias se convierte en un reto y se aconseja que en general será mejor dejar al periférico USB realizar la tarea de adquisición en tiempo real de datos y hacer la comunicación de la tarjeta con el PC host lo menos critica posible de cara al tiempo; es decir realizar grupos de adquisiciones periódicas que forman paquetes de muestras y enviar el paquete al PC host a la velocidad máxima permitida pero con menos frecuencia. La demanda temporal del sistema no es estricta en el sentido de que se determinen totalmente los tiempos de respuesta al evento, sino que se habla de un tiempo de respuesta del sistema expresado en media, por esta razón se considerara sistemas suaves de tiempo real. Además se dan unas recomendaciones para poder obtener menores tiempos de latencia. El uso de sistemas operativos de tiempo real permite la ejecución precisa de tareas en el procesador por medio de asignación de

prioridades. El protocolo utilizado para este proyecto es la versión 2.0 la cual maneja mejores velocidades de transferencia. El autor recomienda también que el PC host que interactúa con el periférico sea un PC dedicado únicamente a la tarea, un sistema embebido (Korver, 2003).

El driver de las DAQ USBDUX está diseñado para su funcionamiento a través de la pila USB del kernel Linux en una versión de USB 2.0. La tarjeta USBDUX utiliza un modo de transferencia isócrona igual que el utilizado en las aplicaciones de reproducción de música o video en tiempo real. El controlador de Comedi ofrece la posibilidad de adquisición/emisión asíncrona y síncrona de señal en una configuración que habilita el modo isócrono de transferencia. A su vez se realizara la instalación de un sistema de tiempo real que regule el sistema de tiempos del sistema y así garantizar unos tiempos de ejecución de las tareas aunque en un modo soft donde no están aseguradas las constantes temporales del sistema de manera estricta pero si en media.

En el proyecto Free and Open Source Software for Industrial Process Control Systems utilizan la tarjeta DAQ USBDUX para la monitorización por medio de la adquisición y gobernar el proceso por medio de la emisión de señales. El método de adquisición empleado en este proyecto es la adquisición síncrona, a su vez se extraen las conclusiones sobre la necesidad de que el sistema debe ser integrado en soft real time. Esto es debido a que no hay elaborada una pila USB que opere en modo hard real time. En este proyecto bajo un sistema operativo Linux 2.6.1X se especifican a frecuencia de muestreo de 1 kHz en el 99% de los casos puede considerarse tiempo real en periodos de 10 ms. La adquisición de señal se realiza de manera síncrona También se refleja en que solo se puede tener comportamiento total hard real time por medio de tarjetas ISA/PCIPC-card I/O. (Mannori et al., 2006)

En la actualidad hay proyectos que han logrado la implementación de un host para USB en versión 2.0 que puede usarse para aplicaciones hard real time sobre sistemas operativos de tiempo real embebidos. Un ejemplo comercial es el RTOS On Time RTOs-32, http://www.on-time.com/rtos-32.htm, donde ofrecen librerías de desarrollo para host PC en tiempo real. En el caso de RTAI existe un proyecto en estatus pre-alpha o alpha en el que se anuncia la creación de una pila USB manejable en hard real time, el proyecto es usb20rt, http://developer.berlios.de/projects/usb20rt/.Este proyecto ofrece una futura implementación de la pila USB 2.0 para Linux/Xenomai. Ofrece la pila del core así como los controladores para el host EHCI, UHCI y OHCI. El proyecto se encuentra en fase de testeo.

En este trabajo, en el contexto de carencia de una pila USB manejable en modo hard real time, se analizan las posibilidades que ofrecen las tarjetas DAQ USBDUX en la implementación de sistema soft real time en RTAI y a través del controlador Comedi. Bajo la precisión temporal ofrecida por RTAI y teniendo en cuenta las limitaciones del protocolo USB se intenta fabricar un mecanismo que permita la emisión/adquisición controlada de señal, es decir una adquisición/emisión de alta precisión, tanto en duración como en momento de actuación. Comedi ofrece la posibilidad de adquisición síncrona y asíncrona, se analizan las posibilidades de ambos modos de interactuación del PC con la tarjeta DAQ. Una vez obtenida esta funcionalidad y comprobada su eficacia, latencia y jitter del sistema se analiza la posibilidad de implementar ciclos cerrados de estimulación dependiente de la actividad.

3 Diseño

En este proyecto y como se anunciaba anteriormente, se ha diseñado un sistema para el control de tarjetas DAQ USBDUX para realizar adquisición/emisión de señales analógicas a través del protocolo USB y con una precisión temporal de pocos milisegundos (gracias a RTAI bajo Linux y mediante las librerías de Comedi). La adquisición de estas señales y la emisión de impulsos para la estimulación se realizaron por medio de una tarjeta DAQ de la familia USBDUX, en concreto con la USBDUX-sigma, una tarjeta de adquisición capaz de manejar señales analógicas de entrada y salida. La tarjeta USBDUX-fast fue utilizada para la monitorización del sistema y por medio de sus mediciones o grabaciones se llevaron a cabo análisis de las latencias medias y jitter del sistema para así poder determinar el grado de precisión del sistema implementado.

Existe una gran variedad de tarjetas de adquisición DAQ en el mercado con grandes prestaciones. La elección de USBDUX para la implementación del sistema se debió a que pueden funcionar bajo el control de driver de carácter open-source integrado en el proyecto Comedi, por lo que estas tarjetas son las únicas DAQ de tipo USB que se pueden utilizar bajo un sistema operativo Linux, y en particular con RTAI. Por otro lado, las prestaciones que ofrecen las DAQs en cuanto al número de canales analógicos o la tasa de muestreo son adecuadas para la realización de este proyecto.

La flexibilidad y dinamismo que ofrece el sistema USB, sistema de comunicación y conexión de periféricos integrados en todos los portátiles, es de gran utilidad en muchos tipos de aplicaciones, ya que facilita el uso de estas DAQ en lugares muy dispares, permitiendo la experimentación in situ. El principal inconveniente que poseen estas tarjetas es que están diseñadas para funcionar a través de la pila USB del kernel Linux, por lo que solo pueden ser utilizadas en un modo temporal soft. La única limitación real de esta tarjeta es el propio protocolo USB, desafortunadamente todavía no es posible utilizar la tarjeta en tareas con condicionamiento temporal estricto (hard realtime), para hacerlo hay que instalar una pila USB del kernel que trabaje bajo un sistema de tiempo real en modo hard. La tarjeta DAQ USBDUX-sigma realiza transferencias isócronas de las muestras que manejan los conversores A/D y D/A. El modo de transferencia isócrono USB es el modo de transferencia que ofrece mejores cualidades para implementar una comunicación entre el PC host y el periférico o tarjeta DAQ USBDUX-sigma en tiempo real. La forma de configuración de la tarjeta DAQ es por medio de comandos enviados en modo de transferencia por bulk, modo no muy aconsejable para la implementación de sistemas en tiempo real pero entre la DAQ y el PC se configura una tubería isócrona que permite crear una tubería entre PC y DAQ con garantías de soft real time. A pesar de este hecho, el resto de características de la tarjeta acredita el uso de tarjetas USBDUX para el desarrollo de este proyecto.

En un primer momento se diseñaron y fabricaron varios conectores, así como los programas para el manejo de las tarjetas DAQ USBDUX en sistemas operativos de propósito general, a través de las funciones ofrecidas por Comedi. Estas conexiones se emplearán para realizar registro en tiempo real de señalización producida desde varias fuentes de señal o estimulación de alta precisión para el control de sistemas, como por ejemplo la neurona artificial. Las conexiones diseñadas utilizan diferentes sistemas de adaptación para poder obtener un amplio rango de señales. Estas señales son generadas desde diferentes fuentes con diferentes características.

Por las razones discutidas en el apartado de RTAI se ha elegido un sistema operativo Linux con extensión de kernel RTAI para permitir que el driver trabaje y ejecute sus funciones en tiempo real soft, es decir se consigue un manejo de la tarjeta que proporciona funcionalidad con restricciones temporal es para lograr una sincronización con el ambiente de actuación.

RTAI y las librerías Comedi se emplean como soporte necesario para crear un sistema que se mantenga en un sincronismo de precisión de milisegundos con el sistema sobre el que se desea experimentar o analizar. La emisión de señales para diferentes sistemas como la neurona artificial desarrollada en el GNB o el registro de la señal de un generador de señales deben realizarse bajo un control temporal estricto con una precisión predefinida.

Gracias a RTAI se obtiene un mecanismo temporal de ejecución de tareas controlado, por lo que podemos realizar estimulación de alta precisión temporal tanto en duración como en momento de actuación. Se comprobará después la eficacia y precisión del sistema por medio de la realización de pruebas de envío de señales periódicas y la implementación de ciclos cerrados de estimulación dependiente de la actividad. La DAQ USBDUX bajo el control de Comedi en un PC con RTAI instalado puede adquirir o emitir los valores de señal que, por ejemplo en el caso de la neurona artificial, determinan estados de excitación, e implementar estimulación dependiente del estado de actividad mediante la tarjeta con restricciones temporales estrictas.

En conclusión, se desea un sistema en soft-real time que realice adquisición/emisión de precisión para enviar estímulos periódicos o realizar ciclos cerrados de estimulación gobernada por objetivo en un PC con sistema operativo RTAI y la tarjeta DAQ USBDUX-sigma controlada por Comedi por medio del protocolo USB. Para demostrar la eficiencia del sistema y la precisión que pueden ofrecer los ciclos cerrados o la estimulación o adquisición de alta precisión temporal se utilizan los comandos de adquisición asíncrona en la tarjeta USBDUX-fast, la cual junto al osciloscopio nos servirán para monitorizar el comportamiento del sistema al completo.

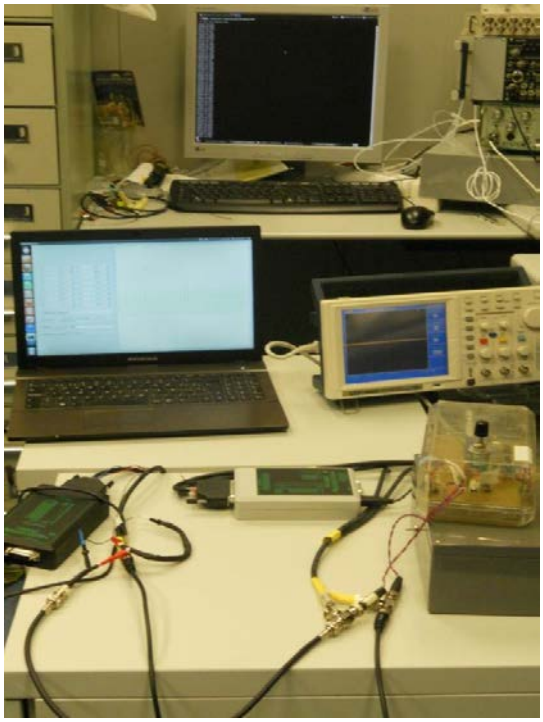


Figura 3: Fotografía del sistema al completo. En orden de arriba abajo y de izquierda a derecha encontramos, el PC con sistema RTAI, encargado de la ejecución de ciclos o funciones de emisión/adquisición de alta precisión, el portátil que almacenara las grabaciones de las señales de interés del sistema (Salida neurona, canales de salida D/A de la USBDUX-sigma), el osciloscopio digital, la DAQ USBDUX-fast que en modo asíncrono realiza adquisición con la precisión que ofrece el hardware de la DAQ, la DAQ USBDUX-sigma, encargada de emisión/adquisición de alta precisión y la neurona artificial, componente donde se analizara los viabilidad del sistema**.**

3.1 Hardware

3.1.1 Tarjeta de adquisición de datos

Para la adquisición de datos se utilizan en este proyecto dos DAQs del tipo USBDUX. Estas tarjetas DAQ están diseñadas para que cualquier maquina con un sistema LINUX (kernel superior a 2.6) y conector USB pueda controlarla. El control y manejo de la tarjeta se realiza en concreto por medio de los controladores (drivers) integrados en Comedi.

Estas tarjetas ofrecen flexibilidad y bajo coste gracias a que están ideadas para entornos open-source y también ofrecen flexibilidad y dinamismo gracias a que se comunican por medio de USB. Las tarjetas DAQ se alimentan vía USB. En concreto están diseñadas y optimizadas para que su bus de datos con el PC sea USB 2.0, el driver necesario para el manejo de las tarjetas DAQ está desarrollado y totalmente adaptado a la pila USB del kernel Linux.

El protocolo USB y la implementación del controlador driver en Comedi ofrece dos formas de configuración del protocolo 2.0: modo full-speed o modo high speed. Estas configuraciones del protocolo ofrecen una latencia fija de transmisión de datos desde el PC a la DAQ de frecuencia máxima de 1 kHz en full-speed o 8 kHz en high speed. Para la emisión síncrona de una muestra simple, la configuración del protocolo USB especifica la latencia media de duración del impulso, de 1 milisegundo o un frame USB para full-speed; en el caso de high-speed es de 500 microsegundos o el tiempo necesario para 4 microframes USB. En la adquisición en modo síncrono, es decir por medio de la adquisición de una muestra analógica simple en un momento conocido, está implementada como un comando de alta precisión idéntico a los utilizados en adquisición asíncrona pero mantiene la tarea bloqueada durante 2 ms. La implementación de la pila USB del kernel Linux obliga a la ejecución en soft real time. Las adquisiciones/emisiones síncronas simples en GPOS ofrecen valores iguales de frecuencia de muestreo o duración de estímulo que en sobre RTAI, la instrucción de adquisición o emisión simple en el controlador de las tarjetas USBDUX-sigma, tarjeta empleada para el sistema en tiempo real, está implementada como un comando de alta precisión idéntico a los utilizados en adquisición asíncrona pero mantiene la tarea bloqueada. El problema es el control del momento de ejecución de la instrucción por parte del sistema operativo. En sistemas de propósito general, la precisión temporal en la ejecución de la acción requerida es peor que sobre sistemas operativos de tiempo real y con mayores e impredecibles tiempos de ejecución.

Ambas tarjetas DAQ USBDUX pueden realizar adquisiciones asíncronas por lo que la tarea de adquisición se realiza en modo background, por medio del protocolo USB en modo de transferencia isócrona y las instrucciones de comando Comedi, una transmisión en la que se garantiza un tasa de transferencia de datos en periodos de tiempos fijados y donde no se producen latencias con jitter. El uso de este método de adquisición permite el diseño y desarrollo aplicaciones multihilos. Se proporcionará más información sobre los modos de adquisición de datos en COMEDI, sección 3.2.2.1 Funciones para interactuación con la tarjeta DAQ.

Ambas tarjetas DAQ cuentan con unos buenos conversores A/D que ofrecen buenos valores de tasas de muestreo. La etapa de adaptación de la señal en cada tarjeta es

diferente. En el caso de la tarjeta DAQ USBDUX-sigma también hay conversores DAC por lo que se puede realizar la emisión síncrona de muestras analógicas de señal. La tarea de estimulación o emisión de señal se realizará sobre esta tarjeta.

3.1.1.1 USBDUX-fast

La DAQ USBDUX-fast tiene 16 canales analógicos de entrada de alta velocidad. Esta tarjeta realiza la adquisición de señales con valores de tensión comprendidas en dos rangos, valores de -0.75 a 0.75 o valores de -0.5 a 0.5. Los rangos de las señales analógicas de entrada pueden ser configurados por software.

La tarjeta cuenta con un circuito integrado de la compañía Cypress, (http://www.cypress.com/), un microcontolador especializado en USB 2.0, modelo CY7C6803A-56PVXC. Este chip es la unidad de control para todos los componentes de la tarjeta, así como el interfaz rápido y eficiente entre el conversor A/D con el puerto USB.

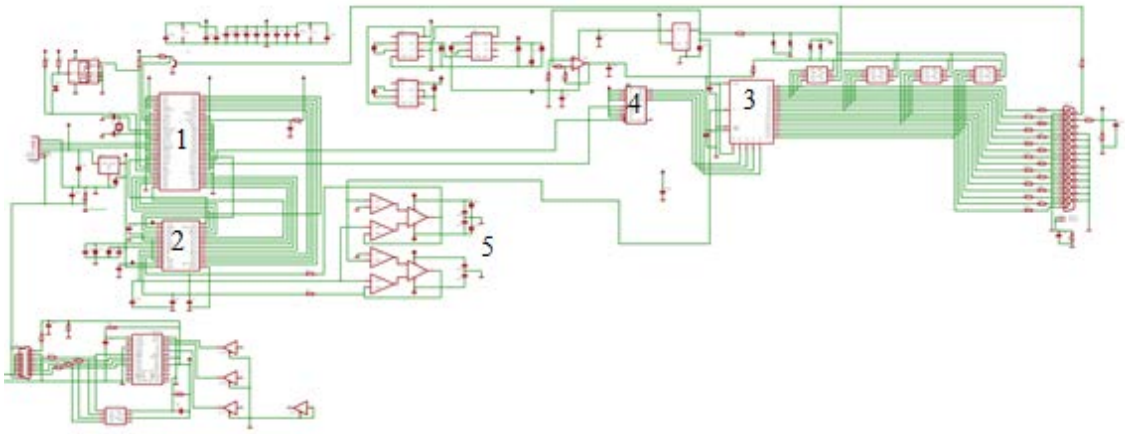


Figura 4: circuito esquemático de la DAQ USBDUX-fast ofrecido por el fabricante. Se observa el microcontrolador especializado en USB [1], los conversores A/D [2], el multiplexor [3], el contador de bits [4] y los amplificadores de instrumentación [5]. En el Anexo: Manual del Programador, 1.2.Esquemático de la DAQ USBDUX-fast se describe en más detalle.

El chip CY7C6803A-56PVXC de la familia EZ-USB-FX2 tiene integrado un microcontrolador 8051, un transmisor USB 2.0 y un SMART SIE (serial interface engine). El micro 8051 tiene una frecuencia de reloj seleccionable de 48, 24 o 12 MHz. La generación de esta señal se realiza por medio de un PLL (Phase Loop) en el que se utiliza como oscilador un reloj externo de frecuencia de oscilación de 24 MHz. El reloj maestro de la tarjeta está fijado en 48 MHz.

Este chip, modelo CY7C6803A-56PVXC, también cuenta con una GPIF(General programmable Interface), que es un interfaz de 8 o 16 pines en paralelo programable por el usuario y permite realizar máquinas de estados finitos que realizan tareas con otros dispositivos ASIC o DSP como el conversor A/D. Otras funciones del GPIF son tareas de control de los diferentes componentes de la tarjeta. Esta interfaz programable permite una comunicación directa entre el transmisor USB y los conversores A/D sin necesidad de que

el microcontrolador 8051 gestione el proceso. Maneja señales de entrada, banderas de listo (RDY) y señales de salida (CNT) para seleccionar el uso de conversor A/D, o como señal de reloj y de Clear para el contador encargado de la demultiplexación del canal de lectura. La implementación del sistema de captación por medio de GPIF provoca otra limitación hardware ya que no es posible realizar multitareas dentro de los tiempos en el que la DAQ tome el control, puesto que se trata de una máquina de estados finitos(véase Designing with the EZ-USB® FX3 Slave FIFO Interface en www.cypress.com)

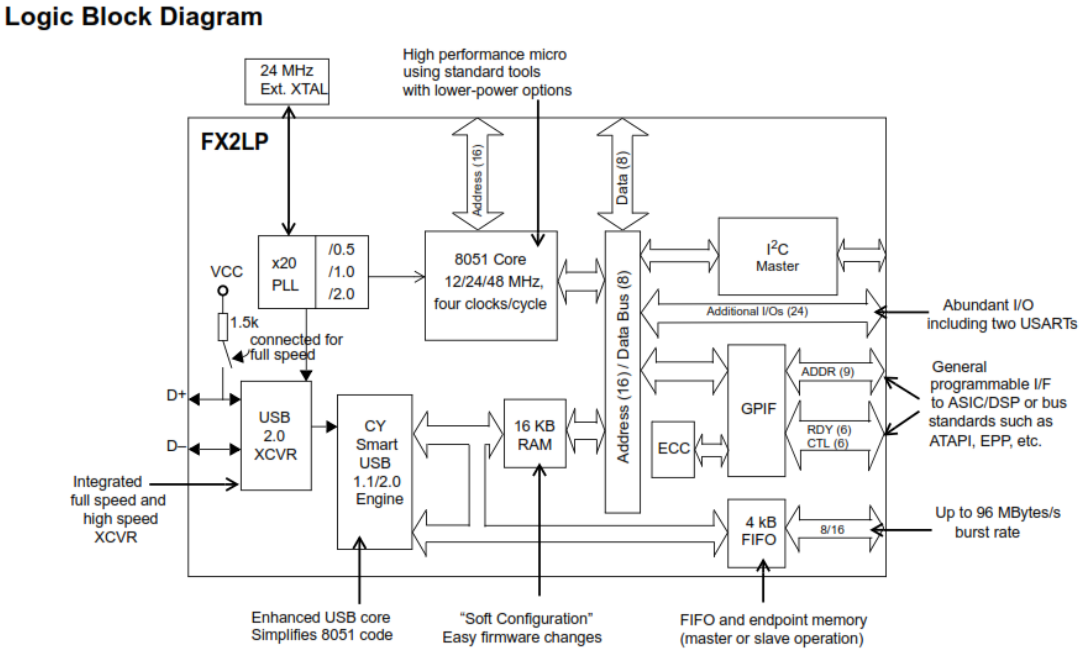


Figura 5: Diagrama de bloques que componen el CY7C6801XX EZ-USB® FX2LP™ USB Microcontroller High-Speed USB Peripheral Controller

La elección de un canal de los 16 posibles canales de lectura se realiza por medio de un multiplexor gobernado por un contador de 4 bits. El contador está sincronizado y controlado por el microcontrolador USB. Esta implementación hardware provoca un límite de tasa máxima de muestreo. Debido a la implementación de selección de canales por medio de un contador de 4 bits, no existe una selección real hardware del canal a la entrada de la tarjeta que se utiliza y, debido a la frecuencia de reloj maestro del sistema de 48 MHz, la máxima velocidad de muestreo ideal alcanzable, teniendo en cuenta solo estas limitaciones hardware es de 3 MHz. El uso de adquisición asíncrona por medio de comando (command) es la solución que ofrece mayores valores de tasa de muestreo.

La señal de canal seleccionado pasa a través de un sistema de amplificadores de instrumentación, etapa en la que se adapta la señal a la entrada de ADC. Los canales de entrada analógicos no tienen buffers a la entrada, la resistencia de estos canales es de 185 kΩ, cuando se conecta una fuente de señal debido a que la entrada de los canales analógicos no tiene un valor de impedancia de entrada grande, para evitar fenómenos de ruido y de reducción del voltaje medido de la señal estudiada, el fabricante recomienda usar sistemas con impedancias de salida muy bajas, por debajo de 50 ohm. La resistencia de entrada del canal junto con el sensor forman un divisor de tensión, la tensión de entrada a la tarjeta está condicionada por los valores de impedancia de entrada del canal y de impedancia de salida del sensor o fuente de señal.

El uso de amplificadores operacionales como seguidores de tensión o buffer es una solución eficiente y barata para poder obtener una buena adaptación entre la DAQ y el sensor o la fuente de señal que se conecta. Gracias a las características del amplificador operacional, la impedancia del voltaje de salida es prácticamente nula, acercando la fuente generadora de la señal a un comportamiento de una fuente de tensión ideal y asilando el componente de sobrecargas o interferencias que se puedan producir desde el componente conectado. La impedancia de salida de una fuente de voltaje ideal es cero. Si la salida no está bufferizada se produce un error de cuantificación. Los canales analógicos de entrada están repartidos en un conector hembra de 44 pines HD (High Definition), ver Tabla 4.

Tabla 4: Asignación de los pines de salida del conector de 44 pin HD hembra de la tarjeta de adquisición USBDUX-fast.

PIN	Dirección	Función
1-14	GND	Tierra
15	Input	Canal A/D 15
16	Input	Canal A/D 0
17	Input	Canal A/D 1
18	Input	Canal A/D 2
19	Input	Canal A/D 3
20	Input	Canal A/D 4
21	Input	Canal A/D 5
22	Input	Canal A/D 6
23	Input	Canal A/D 7
24	Input	Canal A/D 8
25	Input	Canal A/D 9
26	Input	Canal A/D 10
27	Input	Canal A/D 11
28	Input	Canal A/D 12
29	Input	Canal A/D 13
30	Input	Canal A/D 14
31	Input	Canal A/D 15
32	Supply	+ 5 V
33	NC	No conectar
34-44	GND	Tierra

Los conversores analógico digitales convierten la señal analógica condicionada en una muestra digital que el microcontrolador USB recoge, almacena y/o empaqueta en una trama USB para enviarla al PC. Los ADC son de 12 bits de resolución y trabajan a velocidad de megahercios por lo que no representan una limitación en el hardware. La tasa interna máxima de muestreo es de 30 MHz. El interfaz de comunicación o bus con el PC se implementa por medio del puerto USB en configuración USB 2.0. El controlador de la USBDUX-fast y las instrucciones Comedi permiten realizar adquisiciones asíncronas de muestras y transferirlas desde la tarjeta a la PC host por el modo de transferencia isócrono, obteniéndose una garantía alta de envíos temporizados que ofrece posibilidades real time. La USBDUX-fast realizando adquisición asíncrona y volcado de datos adquiridos con

precisión temporal, recepcionándose y procesándose a la frecuencia deseada mediante el programa Comedirecord, constituye un ejemplo de aplicación de registro en tiempo real.

Las características de la tarjeta DAQ USBDUX-fast se pueden resumir en:

- 10 bits resolución máximo nivel de cuantificación 1024
- 16 canales analógicos
- Rango de entrada de señal
    - [-0.5 – 0.5] V
    - [-0.75 – 0.75] V
- Entrada sin buffer a la entrada con resistencia de 185k.
- Necesidad de fuente de baja impedancia, minimiza crosstalk.
- Tasa de muestreo o velocidad de adquisición
    - Modo asíncrono: hasta 3 MHz para un solo canal, cuando hay 16 menos.
    - Modo síncrono: velocidad dependiente de USB speed-mode

A la vista de las características de la tarjeta, USBDUX-fast es un dispositivo muy potente para la monitorización del sistema completo, pudiendo realizar una grabación de la actividad del sistema de estudio y de la emisión de estímulos por parte de la otra tarjeta DAQ USBDUX-sigma.

3.1.1.2 USBDUX-sigma

La USBDUX-sigma comparte la funcionalidad de registro de señales analógicas de la tarjeta DAQ USBDUX-fast aunque el sistema está implementado con otro microcontrolador que ofrece menores tasas de muestreo pero permite la emisión de señal analógica gracias a sus conversores D/A. Este microcontrolador especializado en USB 2.0 de la compañía Cypress es el modelo CY7C6803A-100AXC.

La adquisición/emisión de señales puede realizarse de manera síncrona o asíncrona. Los conversores A/D y la circuitería de entrada están implementados de manera distinta que en la USBDUX-fast y los valores de adquisición asíncrona son de 1 KHz cuando está la configuración de 16 canales muestreando: si son de 3 a 8 canales los utilizados la tasa máxima es de 2 KHz y si es uno o dos canales alcanza tasas de 4 KHz. La adquisición/emisión síncrona depende de la configuración de la velocidad del protocolo USB. La DAQ USBDUX-sigma opera en fast-speed, con latencias de 1 milisegundo o en high-speed, con latencias de 500 microsegundos. En la detección de la tarjeta DAQ se configura el protocolo USB en la velocidad máxima permitida.

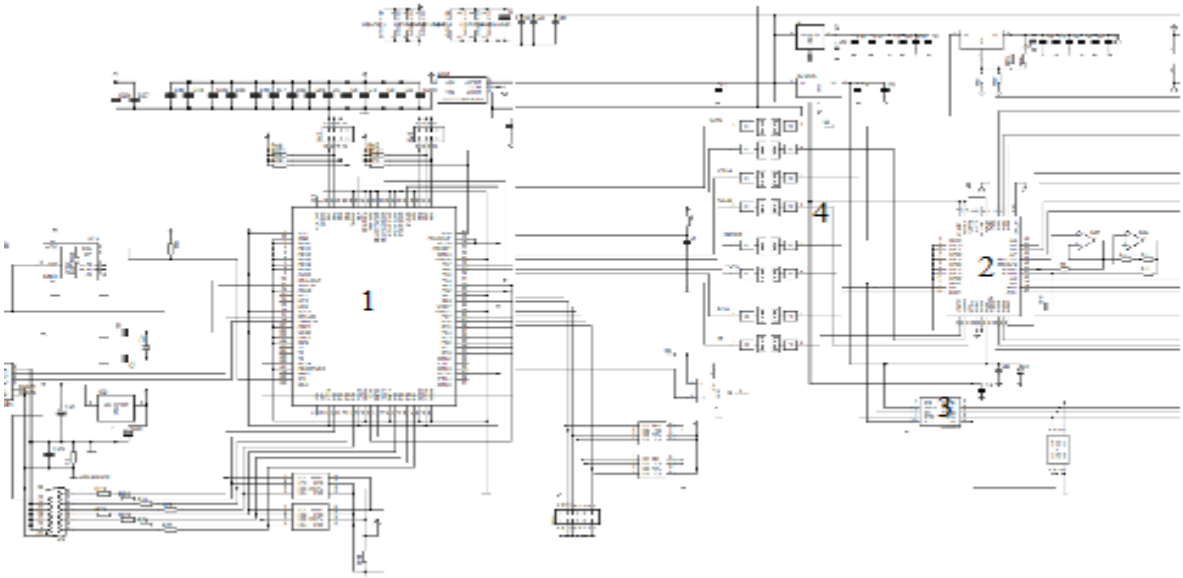


Figura 6: Esquemático parcial de la tarjeta USBDUX-sigma. Se observa el microcontrolador especializado en USB [1], los conversores A/D [2], conversores D/A [3] y transformadores de aislamiento [4]. En el Anexo se encuentra el diagrama a mayor resolución, véase Anexo: Manual del programador, 2.2.Esquemático de la DAQ USBDUX-sigma.

Por el contrario este microcontrolador y la DAQ tienen incorporado el hardware necesario para realizar tareas de estimulación de sistemas, la tarjeta cuenta con conversores D/A y cuatro canales analógicos de salida unipolares para poder emitir señales de tensión de hasta 2.5 V y derivan una corriente máxima de 1 mA.

La tarjeta ofrece una etapa de aislamiento eléctrica para las entradas y salidas analógicas. Por medio de transformadores de aislamiento se logra que no exista una conexión real entre los sensores que conectamos a la tarjeta y el PC que la maneja. Con este sistema de aislamiento el PC queda protegido de las subidas de tensión y se evita los efectos negativos que producen los ground loops (un fenómeno que ocurre cuando se conectan dos aparatos a la misma tierra, punto de referencia de potencial, pero ambos tienen un valor diferente de potencial en ese punto, este fenómeno aumenta considerablemente el ruido en la señal).

La tarjeta DAQ USBDUX-sigma tiene el mismo número de canales analógicos, 16 canales de entrada y mejor resolución en bits que la USBDUX-fast. Los conversores analógico digitales son de 24 bits. La tarjeta USBDUX-sigma también mejora el rango de señal, maneja de -1.325-1.325 V y la resistencia de entrada es de 1 GΩ por lo que las prestaciones respecto a las fuentes de señal mejoran sustancialmente ya que su valor de impedancia no va afectar en el sistema de medida provocando que se reduzca el valor de tensión medido. Para impedancias de salidas altas en sistemas que se desean medir, se produce un error en la tensión por reducción de la corriente que llega al conversor analógico digital. La entrada en este caso está bufferizada. Utiliza también un conector de tipo 44 pin High Density y la distribución de los pines y su funcionalidad están detalladas en la Tabla 5.

PIN	Dirección	Función
1-14	GND	Tierra
15	Input	Canal A/D 15
16	Input	Canal A/D 0
17	Input	Canal A/D 1
18	Input	Canal A/D 2
19	Input	Canal A/D 3
20	Input	Canal A/D 4
21	Input	Canal A/D 5
22	Input	Canal A/D 6
23	Input	Canal A/D 7
24	Input	Canal A/D 8
25	Input	Canal A/D 9
26	Input	Canal A/D 10
27	Input	Canal A/D 11
28	Input	Canal A/D 12
29	Input	Canal A/D 13
30	Input	Canal A/D 14
31	Supply	-5 V
32	Supply	+ 5 V
33	NC	No conectar
34-40	GND	Tierra
41	Output	Canal D/A 0
42	Output	Canal D/A 1
43	Output	Canal D/A 2
44	Output	Canal D/A 3

Tabla 5: Asignación de pines y funcionalidad conector HD-44 pin para la DAQ USBDUX-sigma.

El protocolo USB otorga flexibilidad, y la tarjeta puede funcionar en modo asíncrono realizando trasferencias de muestras en modo isócrono, en este escenario el reloj de la tarjeta impone las condiciones temporales y por lo tanto el sincronismo del sistema. Si el sincronismo entre la tarjeta DAQ y el PC host lo impone el sistema de temporización del PC, en Comedi encontramos funciones simples de adquisición o emisión de una muestra. En una configuración en full-speed en el controlador de las tarjetas DAQ USBDUX-sigma la transferencia de muestras se realiza en frames USB, lo que limita la tasa de ejecución de las instrucciones a un tiempo de latencia fijo. En emisión, la tasa máxima es de 1 KHz, y en lectura de 500 Hz. Es decir, el tiempo mínimo de emisión de una muestra de señal analógica para la monitorización o control de un sistema es de uno y dos milisegundos en configuración full-speed. En modo high-speed se obtienen valores de microsegundos, 500 μs.

El driver controlador ofrecido por Comedi utiliza la pila USB del kernel Linux. Esto es una limitación considerable ya que no se está implementando un sistema en tiempo real hard estricto en el que se pueda asegurar que no ocurrirá la perdida de una señal y por tanto que se pueda provocar el colapso del sistema. También la temporización del sistema no es del todo estricta. La emisión o recepción de señales es controlada por el kernel, el cual debido a la implementación de la pila USB no puede ejecutarse en hard real time.

Otra mejora respecto a la USBDUX-fast es que la USBDUX-sigma incorpora la funcionalidad de emisión de señales para poder controlar o enviar señales de estímulo a otros sistemas. La tarjeta USBDUX-sigma puede realizar la funcionalidad de ciclo cerrado de estimulación dependiente de la actividad, ya que tiene la capacidad de monitorizar el sistema por medio de los canales analógicos de entrada y a su vez puede ejecutar la emisión precisa y controlada por medio de los canales analógicos de salida. Esta tarjeta incorpora una mejora de ejecución de las instrucciones síncronas de lectura consiguiendo en sistemas operativos de propósito general tiempos de ejecución de 2 ms como mínimo es decir 500 Hz de tasas de muestreo máxima. En configuración asíncrona las tasas son superiores. La emisión de señales por tanto está limitada a una frecuencia máxima y trabaja en un rango unipolar donde el voltaje máximo es de 2.5 V. Son utilizados 256 niveles de cuantificación para la emisión de señal. El error de cuantificación que manejamos es de casi 5 mV.

La tarjeta incorpora también 24 pines de entrada digital programables por medio de Comedi y también puede ser configurada para realizar funciones de PWM (pulse wave modulation).Las características de la tarjeta DAQ USBDUX-sigma se pueden resumir en:

- Conversores A/D
    - 24 bits de resolución
    - 16 canales analógicos
    - Rango de entrada de señal
        - [-1.325 – 1.325] V
    - Resistencia de entrada > 1G.
    - Tasa de muestreo o velocidad de adquisición
        - Modo asíncrono: hasta 4 kHz un o dos canales ,2 kHz de 3 a 8 canales y 1kHz si se utilizan los 16.
        - Modo síncrono: 2 milisegundos implementado como comando.
- Conversores D/A
    - 8 bits de resolución
    - 4 canales analógicos
    - Rango de entrada de señal unipolar:
        - [0 - 2.5] V
    - Máxima corriente derivada 1 mA.
    - Tasa de muestreo o velocidad de adquisición
        - Modo asíncrono: hasta 4 kHz un o dos canales ,2 kHz de 3 a 8 canales y 1kHz si se utilizan los 16. Tiempo de reacción de un canal 250 microsegundos (2 USB frame) high-speed.
        - Modo síncrono. Depende de configuración USB.

3.1.2 Conexiones

Las tarjetas DAQ USBDUX se conectan al PC por medio de USB. Las tarjetas DAQ USBDUX se conectan a dispositivos externos por medio de un conector de 44 pines HD (high density). En este proyecto se ha diseñado y construido una conexión genérica para cada tarjeta. A través de esta conexión se conectará la tarjeta a las diferentes fuentes de señal que queramos medir o analizar, las cuales como en el caso de algunos sensores ofrecen niveles de potencia de señal muy bajos por lo que el aislamiento al ruido y a los fenómenos de interferencia serán principios de diseño de las conexiones.

Como ya se ha comentado a lo largo de la memoria, las tarjetas DAQ están diseñadas para ser utilizadas en diversos entornos ya que por sus prestaciones ofrecen gran versatilidad. Por esa razón la estabilidad mecánica, la flexibilidad y la fiabilidad son también principios de diseño. Las conexiones tienen una longitud de 1.5 m, esto conlleva que se puedan endurecer los efectos de interferencia entre los canales. El aislamiento y apantallamiento se reflejan de nuevo como principios de diseño. Por todas estas razones se ha elegido un cable de sonido comercial que ofrece unas características muy buenas contra el ruido electromagnético exterior y que brinda un asilamiento entre los canales. A su vez ofrece una gran resistencia y flexibilidad, estos cables están diseñados para micrófonos y conexiones de instrumentos. Los fenómenos de *crosstalk* que puedan acontecer a lo largo del cable de conexión están muy limitados.

La tarjeta será empleada tanto para la monitorización como para la emisión de pulsos que modifican el estado de excitación de una neurona artificial. La neurona artificial tiene un conector BNC hembra. El uso de este tipo de conectores es una práctica muy extendida por lo que la entrada a los canales se realiza con un conector BNC macho de 75 Ohm. Se ha diseñado una conexión o cable para cada tarjeta de adquisición ya que cada tarjeta tiene diferente funcionalidad y finalidad.

3.1.2.1 Conexión para la USBDUX-fast

La tarjeta USBDUX-fast está diseña para la adquisición de señales analógicas y solo posee canales analógicos de lectura. Se ha diseñado un cable de dos canales analógicos, un canal con conector BNC macho y otro canal con conector banana, con el cable de sonido de dos núcleos. Este cable se conecta a la tarjeta de adquisición por medio del conector de 44 pines HD, del cual son utilizados 2 canales analógicos. También para futuras aplicaciones se habilitan cables para los pines de Supply (alimentación de 5 voltios) y el pin de entrada de reloj externo. Los pines seleccionados para ser cableados no deben quedarse en circuito abierto cuando no se estén utilizando, es necesario conectarlos a tierra para evitar fenómenos de *crosstalk* y que aumente el nivel de ruido del circuito. Los canales que no son seleccionados también son conectados a tierra en el propio conector para reducir fenómenos de *crosstalk* y reducir el ruido total del sistema.

3.1.2.2 Conexión para la USBDUX-sigma

El cable diseñado para la tarjeta USBDUX-sigma cumple con las mismas especificaciones que el cable de la USBDUX-fast en cuanto a la adquisición de señal analógica por lo que está dotado de dos canales analógicos. La DAQ USBDUX-sigma incorpora funcionalidad de emisión de señal.

El cable USBDUX-sigma será empleado para la monitorización y estimulación de un sistema, por ejemplo la neurona artificial, por lo que se ha diseñado con dos canales de salida para la emisión de señales analógicas y dos canales analógicos de lectura.

Se han empleado dos cables de sonido de 1.5 metros de longitud para la implementación de los 4 canales necesarios. Los conectores empleados son BNC para la conexión con las diferentes fuentes de señal y con la tarjeta por medio del conector 44 pin HD.

3.1.3 Adaptación de señal

Los sensores o fuentes de señal que monitoriza o controla la DAQ trabajan en rangos de señal muy dispares. Las características de señal necesarias para que la tarjeta pueda adquirirla se obtienen en una fase de adaptación de la señal. La tarjeta DAQ trabaja con unos rangos determinados configurables por sotfware. Una buena fase de adaptación de la señal otorga mejor precisión y características de adquisición de señal. En la guía para construir un sistema de medidas ofrecida por National Instrument encontramos una visión de los fenómenos físicos que hay que atender para lograr una adaptación de señal adecuada. A continuación se detallan los métodos de adaptación de la señal empleados o valorados en las DAQ USBDUX:

- Atenuación :

Los rangos de tensión de las entradas analógicas están limitados a señales bipolares de 0.75 V ó 0.5 V en el caso de la USBDUX-fast y en 1.35 V en la USBDUX-sigma. Cuando se conectan fuentes de señal con valores de tensión alta, señales de amplitud de 10V como en el caso de la neurona artificial, es necesario atenuar la amplitud de la señal para poder adquirir la señal.

- Amplificación :

Es el caso contrario a la atenuación, cuando trabajamos con sensores o fuentes de señal que emiten señales de amplitud del orden de los mV es necesario aumentar el valor de amplitud de la señal y protegerla ante el ruido. El ruido es una constante en todos los sistemas, resulta un problema mínimo si se consigue que la señal se emita en buenos rangos de señal, varios ordenes mayores que el valor de tensión que represente el ruido.

- Aislamiento:

La tarjeta USBDUX-fast no ofrece ningún sistema de aislamiento entre la tarjeta y el ordenador por lo que existe una conexión eléctrica real entre las fuentes de señal o sensores y el PC que maneja la DAQ. Esto es un problema a tener en cuenta ya que pueden ocurrir fenómenos de ground looop o fenómenos de sobrecarga de tensión que puedan acontecer tanto desde las fuentes al conector USB como desde el PC a los sensores conectados los

canales. Si se va a utilizar la tarjeta para la adquisición de señal biológica, en la conexión entre la tarjeta y los sensores se debe realizar algún circuito que proteja a la tarjeta y al PC de fenómenos eléctricos indeseados, y viceversa. La colocación de un buffer a la entrada no desconecta eléctricamente el circuito pero sí que le protege de sobrecargas de tensión. La tarjeta USBDUX-sigma está equipada con una etapa de aislamiento, una vez realizada la digitalización de la muestra analógica recogida es conducida por un transformador de asilamiento.

- Filtrado:

Con el filtrado se consigue que el ruido producido por interferencias a altas frecuencias desaparezca. La realización de filtros paso bajo sirve para eliminar señales de ruido de alta frecuencia tales como las producidas por las fuentes de alimentación conectadas a la red eléctrica, ruido eléctrico que ocurre entrono a 50 o 60 Hz. Debe realizarse filtrado antialiasing para reducir el ancho de anda de la señal emitida y así mejorar la SNR del sistema.

3.1.4 La neurona artificial

En el GNB se ha desarrollado una neurona artificial que simula el comportamiento de una neurona real. Durante el desarrollado de un software de dynamic clamp avanzado en su fase de testeo y evaluación se utilizó una neurona artificial en tiempo real que se basa en las ecuaciones diferenciales del modelo matemático de Hindmarsh-Rose (Muñiz, 2005; Muniz et al., 2009).

$$\frac{dx(t)}{dt} = ay(t) + bx^2(t) - cx^3 - dz(t) + \epsilon + Isyn$$

$$\frac{dy(t)}{dt} = e - fx^2(t) - y(t)$$

$$\frac{1}{\mu}\frac{dz(t)}{dt} = z(t) + S[x(t) + g]$$

Estas ecuaciones determinan el potencial de la membrana *x* y los efectos derivados de la combinación de corrientes de iones rápidas y lentas (*y*, *z*). No es objetivo de este proyecto el conocer la dinámica de la neurona sino más bien probar sobre ella la efectividad del sistema de adquisición/emisión, y en particular del ciclo cerrado.

La neurona tiene una salida analógica donde se representa el potencial de la membrana, también está dotada de una entrada analógica y de un potenciómetro, ambos componentes pueden ser utilizados para derivar corriente externa en el sistema y así provocar que la neurona entre en otro régimen de excitación.

Los regímenes o estados de excitación que podemos detectar en la neurona son regímenes en forma de spikes (potenciales de acción) o estados en forma de burst o grupo de spikes (ráfagas). A continuación se muestran estos dos tipos de regímenes que podemos registrar en la neurona.

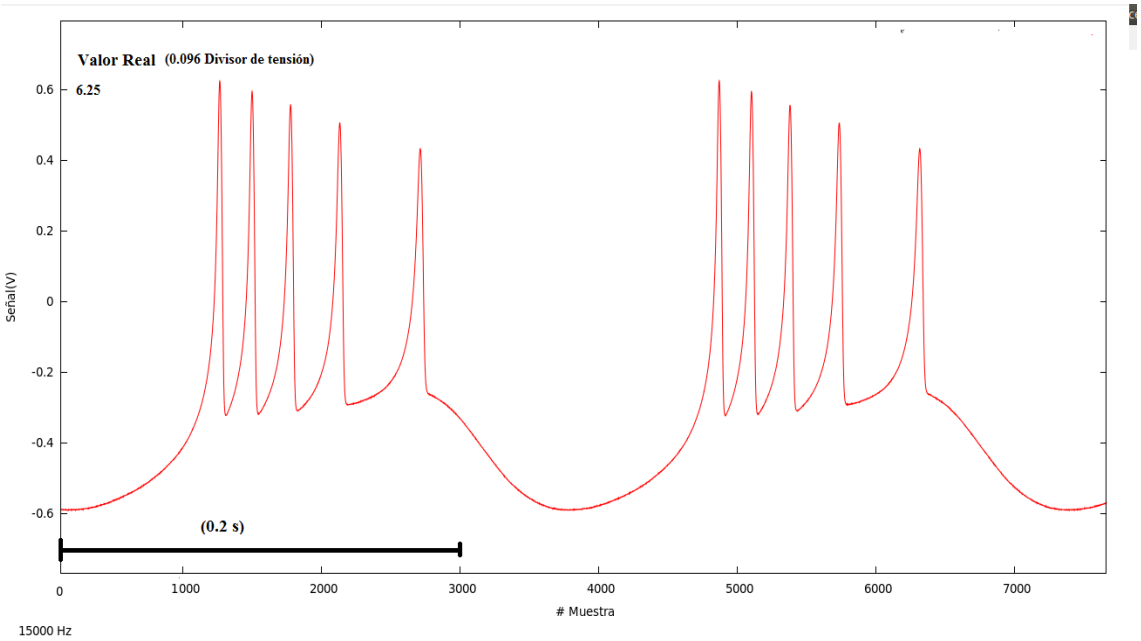


Figura 7: Grabación realizada con la USBDUX-fast de la neurona artificial en estado de actividad en forma de burst o grupos de spikes. Se emplea un divisor de tensión de aproximadamente 0.1.

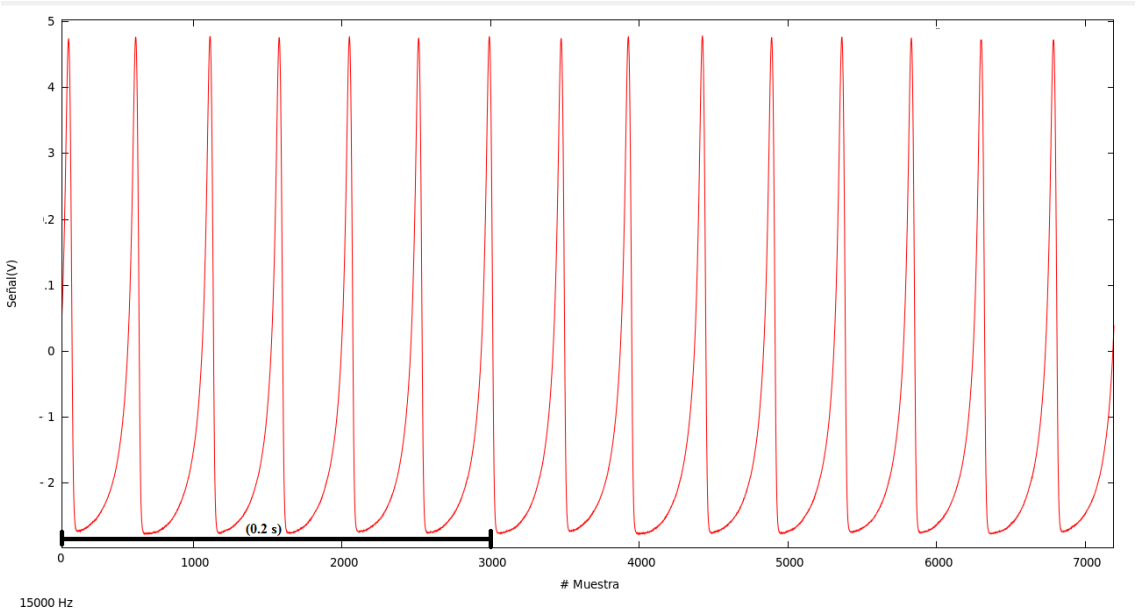


Figura 8: Representación de la grabación de la neurona artificial en estado de emisión actividad spiking. Se aplica corrección del divisor de tensión en este caso.

Para realizar un análisis de la precisión de la manipulación de señal analógicas por parte de las DAQ USBDUX se analiza sobre la neurona la viabilidad de la ejecución de ciclos cerrados. Para ello se realizan dos conexiones a la neurona, por una parte se conecta un canal analógico de la DAQ USBDUX-sigma para obtener lecturas del potencial que representan estados de excitación de las neuronas, y por otra parte se le conecta a un canal

analógico de emisión. El PC envía una muestra digital a la tarjeta DAQ USBDUX-sigma la cual emite señales que inyectan corriente a la neurona controlando o estimulando a la neurona, perturbando el estado de la misma.

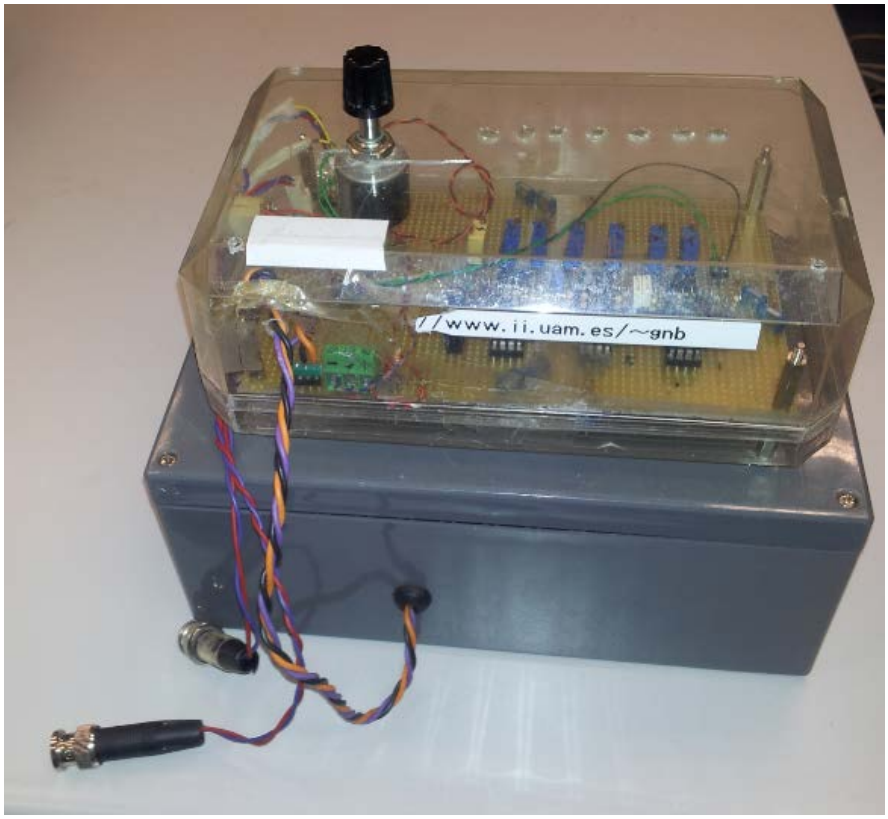


Figura 9: Foto de la neurona artificial desarrollada en el laboratorio del GNB y utilizada en este proyecto para la estimulación de alta precisión temporal y comprobación de la implementación de ciclos cerrados.

3.2 Sotfware

En esta sección se comenta las características software del diseño. Estas características difieren dependiendo del tipo de sistema operativo que tome control del PC. Las tarjetas DAQ están diseñadas para funcionar bajo sistemas operativos de propósito general (GPOS) bajo el control de la pila USB de un kernel Linux. Para poder dotar al PC de un dominio temporal sobre la tarjeta DAQ, y así realizar el registro o control de señales en tiempo real, es necesario utilizar un sistema operativo en tiempo real. En concreto, en este proyecto se utiliza RTAI, y por medio de los mecanismos de control de las tareas o procesos y la posibilidad de implementación de tareas periódicas que realizan peticiones síncronas sobre la tarjeta DAQ, se podrá abordar la construcción de un sistema de ciclo cerrado en soft real time. El tiempo real es soft debido a que la DAQ transfiere los datos y se comunica con el PC por medio del protocolo USB. Los controladores están diseñados para ser utilizados sobre la pila USB de un kernel Linux, estas funciones necesitan del sistema operativo de propósito general para realizar su actuación. Esto provoca que sea posible la intrusión de otro tipo de peticiones de sistema operativo en el momento de necesidad de utilización de la CPU provocando *jitter* sobre la latencia del sistema.

3.2.1 GPOS (General Purpose Operation systems)

3.2.1.1 LINUX

Las drivers que manejan las tarjetas DAQ están en COMEDI, proyecto de open-source perteneciente al mundo Linux. Este tipo de sistemas operativos ofrece gran dinamismo en desarrollo, su kernel o código fuente del sistema operativo está escrito en C. Además del kernel, los sistemas operativos basados en UNIX ofrecen entornos de desarrollo, librerías de funciones o comandos, todos ellos modificables en su código fuente. El fácil acceso y manipulación de todo tipo de componentes que engloban el sistema operativo, así como que los drivers de manejo de la tarjeta estén desarrollados para funcionar bajo un sistema LINUX, convierte a cualquier distribución con kernel LINUX superior a 2.6 en una solución viable.

Los kernel Linux y los drivers de COMEDI están comercializados bajo GPU general public license, por lo que podremos modificar, copiar y distribuir todo el código con el que trabajamos en el proyecto. La distribución de Linux elegida en concreto es Ubuntu. Esta distribución tiene un kernel LINUX y a su vez ofrece una gran variedad de funcionalidad al usuario de manera muy sencilla e intuitiva. Ubuntu otorga la posibilidad de instalación gratuita de una gran cantidad de herramientas de desarrollo que no necesitan costosas licencias. También es una de las distribuciones LINUX más populares (ver https://gnu.org/ /gpl.html).

Ubuntu como sistema LINUX es un sistema basado en una arquitectura monolítica en la que todo recurso tiene asociado un fichero. El sistema operativo cuenta con una serie de archivos, drivers para el manejo de los componentes hardware y demás tipos de objetos que son unidos de manera estática al inicializar el sistema operativo que carga estos recursos necesarios para el normal funcionamiento del sistema. También cuenta con la posibilidad de cargar módulos de manera dinámica y así insertar nuevas funcionalidades por medio de tareas o módulos al sistema operativo Linux.

3.2.2 COMEDI

Comedi es un proyecto de software libre que desarrolla controladores para tarjetas DAQ, herramientas y librerías para varias formas de adquisición, lectura y escritura analógica o digital, generación de pulsos, etc. El manual *Control and Measurement Device Interface* handbook for Comedilib 0.10.1 describe la funcionallidad de estas librerías.

Comedi realiza la función de interfaz entre la tarjeta y el sistema operativo, este mecanismo de interfaz se realiza por la combinación de un API de funciones genéricas independientes a qué tarjeta maneje, de una colección de controladores o drivers para un amplio rango de tarjetas y una librería de funciones para su uso y manejo. Con estos componentes se pueden desarrollar aplicaciones multipropósito con las tarjetas.

El proyecto Comedi distribuye su código en varios paquetes:

- Comedi es la colección de los controladores para un amplio número de tarjetas o devices según la terminología de Comedi. La funcionalidad del controlador o driver se realiza por medio de la combinación de dos módulos del sistema, el módulo Comedi y uno específico a la tarjeta que maneje.

- Comedilib es la librería en espacio usuario que contiene el API de funciones para la manipulación de las tarjetas. Establece un interfaz para el desarrollo de aplicaciones en espacio usuario y realiza una comunicación con los módulos de Comedi que actúan a espacio kernel y manipula y gobierna el hardware de sistema.

- Kcomedilib. Una de las características más destacables de Comedi es que puede ser integrada y utilizada en sistemas operativos de tiempo real como RTAI. Kcomedilib es la librería que realiza la función de interfaz amigable para el desarrollo de aplicaciones sobre tareas de tiempo real en espacio kernel de manera similar a la función realizada por Comedilib pero con pérdida de funcionalidad, por ejemplo las funciones de conversión de datos a valores de señal analógica no son aplicables en modo kernel. Esta librería también debe introducirse como módulo, siendo necesario para poder usar la tarjeta DAQ USBDUX en un sistema operativo en tiempo real como RTAI.

El proyecto Comedi solo ofrece la parte mecánica del periférico que controla, no ofrece una política de uso concreto para la tarjeta o el periférico, se da únicamente una funcionalidad básica, como la emisión de pulsos o la adquisición de muestras analógicas. El algoritmo de control, los mecanismos de proyección de los datos de manera gráfica o las librerías de procesado de señales, tienen que ser implementados a través de programas dedicados.

Por ultimo cabe resaltar la organización y notación del hardware que se realiza en Comedi:

- *Canal* es la mínima unidad hardware en Comedi, representa un simple canal de datos, por ejemplo canal analógico de entrada o salida.

- *Subdevice* es un conjunto de canales con funcionalidad común, por ejemplo 16 canales analógicos de entrada componen el subdevice 0 y 4 canales de salida analógica se agrupan en el subdevice 1.

- *Device* es la tarjeta DAQ en sí, en la que se recogen varios subdevices. Por ejemplo la DAQ USBDUX-sigma.

Comedi es capaz de manejar señales digitales y analógicas o señales basadas en pulsos, como señales de reloj. Solo se utilizan señales analógicas en este proyecto.

3.2.2.1 Funciones para interactuación con la tarjeta DAQ

Los diferentes métodos que ofrece Comedi para el manejo de un canal de la tarjeta DAQ son las siguientes:

- Funciones para adquisición simple.
- Instrucciones para adquisiciones múltiples.
- Comandos para la adquisición asíncrona.

En la terminología de Comedi se utiliza el término adquisición para referirse a cualquier tipo de interactuación con la tarjeta o device, es decir no solamente se refiere a la adquisición simple de una muestra sino también a una estimulación concreta o la emisión de señal analógica. Las funciones que realizan estas tareas son comedi_data_read y comedi_data_write. La función de adquisiciones múltiples realiza la misma tarea que las funciones de adquisición simple por lo que mantiene bloqueado el proceso hasta que se realiza la acción completa de adquisición pero no garantiza una precisión temporal por hardware entre cada adquisición. Por esta razón, la presión temporal del sistema se ve afectada y no se utiliza para este proyecto.

Los comandos para la adquisición asíncrona logran mayores tasas de muestreo debido a que son las tarjetas DAQ las que imponen las condiciones temporales para la adquisición. Por medio del comando se realiza la tarea periódica de adquisición en espacios de tiempo llamados scan, por lo que una vez se han configurado los parámetros del command o comando en la tarjeta, esta actúa de manera autónoma, sin un control temporal desde el PC sobre la tarjeta, pudiéndose realizar más acciones en la aplicación que esperar por los datos. Esto provoca que la adquisición asíncrona por sí sola no sea la manera más adecuada para realizar ciclos cerrados de estimulación dependientes de la actividad o conducidas por objetivo. También es necesaria una precisión temporal de actuación y un control por parte de un reloj global del sistema que sincroniza todas las tareas, tanto de monitorización como de estimulación, que componen el ciclo cerrado.

El volcado de datos por la tarjeta DAQ USBDUX al PC se realiza por medio del protocolo USB en modo de transferencia isócrono, por lo que se asegura un mecanismo temporal que permite determinar las latencias del sistema. Este modo de transferencia es el utilizado para la reproducción de música o video desde periféricos USB en tiempo real no estricto ya que pueden ocurrir errores. Este modo de transferencia del protocolo USB habilita un canal de comunicación entre el PC y la tarjeta DAQ USBDUX con garantías de envío o realización de transferencia en tiempo pero sin control de errores, es decir los datos llegan a un ritmo

constante pero debe tolerarse fallos. Esta es la forma de enviar grandes cantidades de datos a través de un bus ocupado (Axelson, 2005).

Las tarjetas DAQ USBDUX manejan estos comandos ofreciendo microsegundos de precisión por lo que se puede utilizar para la tarea de monitorización del sistema de experimentación. Sin embargo para el envío de señales plantea más problemas ya que si la tarjeta DAQ está realizando una secuencia de scan constante de manera asíncrona, no hay forma de interactuar con la tarjeta y, al detectarse un espacio temporal concreto de necesidad de emisión de señal de control, la tarjeta se encuentra ocupada realizando la tarea de monitorización por lo que la emisión de una señal analógica de salida o estimulo no es posible en el tiempo de actuación necesario.

Observando el código del controlador de la tarjeta DAQ USBDUX-sigma la simple emisión de una muestra por medio del API de Comedi, comedi_data_write se realiza por medio de una secuencia de scan o command en una tarea con una latencia fija dependiente de la configuración de la velocidad de transferencia, 1 milisegundo o frame en full-speed y de 500 microsegundos en high-speed. En el caso de la instrucción simple de lectura de una muestra o adquisición se necesita dos frames USB para realizar la instrucción por lo que la precisión de muestreo está en 2 ms muestra o 500 Hz. Por medio de las instrucciones simples de emisión/adquisición se obtiene un mecanismo de sincronización que permite que el host PC tome la responsabilidad temporal de la ejecución de instrucciones en tiempos concretos. Esta es una opción para este proyecto, llamada configuración síncrona, la implementación de un sistema gobernado por RTAI y la ejecución controlada por las herramientas de temporización del sistema operativo en tiempo real de las instrucciones simples de adquisición ofrecidas en Comedi y gracias a la implementación de estas funciones con el controlador driver de la DAQ USBDUX-sigma.

Los comandos o secuencias de muestreo en aplicaciones simples que no requieran de ciclos cerrados en tiempo real, sino que operen en un único sentido como aplicaciones de monitorización o grabación constituyen una herramienta muy potente, ya que es la tarjeta la que vuelca la información trabajando de manera independiente y estudiada a un ancho de banda asignado y permitiendo exprimir así todas las posibilidades hardware que las tarjetas DAQ poseen. Esta configuración es llamada asíncrona. La información es almacenada o procesada al ritmo posible por el PC host obteniéndose un mecanismo con control temporal. La USBDUX-fast está diseñada para poder usar los command por lo que será la tarjeta que se utilice para la monitorización del sistema de tiempo real el cual es implementado en un PC con RTAI instalado y con la tarjeta USBDUX-sigma realizando las tareas de adquisición/emisión síncrona o asíncrona de señal analógica. Con las grabaciones realizadas evaluaremos las características y límites de señal emitida por la DAQ USBDUX-sigma.

En resumen la adquisición o estimulación síncrona se logra por medio un reloj maestro que gobierna o controla los tiempos del sistema en global, tanto el PC como la tarjeta DAQ USBDUX-sigma y marca la ejecución de las diferentes tareas las cuales por medio de funciones de simple adquisición o estimulación realizan los ciclos cerrados de estimulación dirigidos por objetivo. En la adquisición asíncrona la temporización está dada por la tarjeta, la cual se configura a una frecuencia de trabajo y el volcado de muestras se realiza de manera isócrona a través de USB. El uso de los datos de manera controlada, y la posibilidad de manipulación única muestra y en tiempos adecuados, posibilita el

comportamiento en tiempo real. En Kcomedilib encontramos un mecanismo de sincronización para el sistema en configuración asíncrona.

3.2.2.1.1 Adquisición o estimulación de una muestra simple.

Comedi ofrece la posibilidad de realizar una llamada a un canal para realizar de manera sincronizada una sola actuación sobre la tarjeta DAQ, esta función es llamada adquisición simple pero se refiere también a estimulación simple tanto en analógico como con pines digitales. Este sincronismo se logra por medio del bloqueo del proceso que realiza la petición hasta que se realice la acción simple requerida.

La tarjeta USBDUX-sigma ofrece un tiempo mínimo de latencia de 500 microsegundos en modo high-speed, es decir la tarjeta bloquea el proceso ese periodo de ejecución. Esta precisión temporal se logra por la implementación de un comando de precisión. Las funciones de Comedi para simple adquisición y estimulación analógica en espacio usuario son comedi_data_read y comedi_data_write. Como las tarjetas DAQ se utilizan en soft real-time y con RTAI que permite trabajar con funciones de espacio usuario de los sistemas operativos de propósito general, se utilizarán estas mismas funciones para las tareas RTAI. Este modo de actuación síncrona entre la tarjeta DAQ y el PC permite un soporte para la realización de estimulación de alta precisión temporal en el rango de milisegundos.

Ambas funciones necesitan de un rango de referencia para caracterizar la medición, por lo que es aconsejable una vez realizada la configuración de la tarjeta realizar una llamada a comedi_test, que proporciona todas las características del device, subdevice y canales, sus rangos de referencia, que en el caso de nuestras tarjetas DAQ son rangos de tensión, tipos de referencia analógica que utiliza. Para este proyecto la referencia se fija en tierra. Los datos que manejan estas funciones son una estructura llamada sampl_t o en su versión más grande lsampl_t. Este valor puede transformarse a unidades físicas gracias a los rangos de los canales y al máximo dato que manejan los conversores A/D. Cada canal tiene una estructura de datos, comedi_range, en la que queda guardado el valor mínimo y máximo de la tensión con la que trabaja el canal.

Comedilib ofrece funciones de conversión de lsampl_t a valor físico o al revés. En el caso de RTAI esta función no se encuentra en el API de Kcomedilib.

3.2.2.2 Adquisición asíncrona Command

Los comandos son la herramienta que más potencia las características de adquisición de las tarjetas ya que el proceso de adquisición se realiza bajo el dominio temporal de la tarjeta no del que se impone desde el PC. Las tasas de muestreo dependen de la implementación hardware de cada tarjeta.

El comando *command* especifica una secuencia de datos a adquirir/emitir concreta, consiste en un número de scans periódicos, en el que en cada scan se realiza una serie de conversiones de valores analógicos en muestras o viceversa. El número de conversiones depende del número de canales que se desean utilizar.

En la siguiente figura disponible en la documentación de Comedi se explica cómo se realiza la adquisición asíncrona.

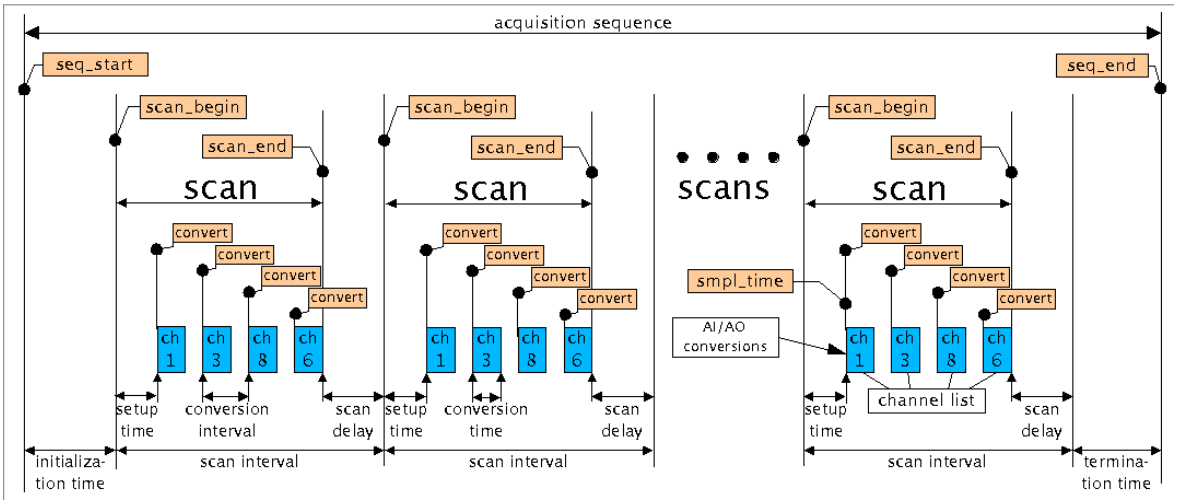


Figura 10: Esquemático temporal de la ejecución de un command o comando Comedi, la adquisición asíncrona se realiza por medio de secuencias periódicas idénticas, scans, en los que se realiza la conversión de las muestras obtenidas en los canales configurados.

Como se puede observar la secuencia tiene unos límites temporales fijados, en esta secuencia se repiten de manera periódica, scan interval, el escaneo y conversión de un número determinado de canales. Esta conversión está separada temporalmente por un parámetro llamado convertion time, donde se obtiene un sampl_t(muestra) de la conversión de una muestra analógica. La funcionalidad de los comandos se fija en varios campos de una estructura, comedi_cmd:

```
typedef struct comedi_cmd_struct comedi_cmd;

struct comedi_cmd_struct {
  unsigned int subdev;          // which subdevice to sample
  unsigned int flags;           // encode some configuration
possibilities
                                // of the command execution; e.g.,
                                // whether a callback routine is to
be
                                // called at the end of the command

  unsigned int start_src;       // event to make the acquisition
start
  unsigned int start_arg;       // parameters that influence this
start

  unsigned int scan_begin_src; // event to make a particular scan
start
  unsigned int scan_begin_arg; // parameters that influence this
start`
```

```
  unsigned int convert_src;    // event to make a particular
conversion start
  unsigned int convert_arg;    // parameters that influence this
start

  unsigned int scan_end_src;   // event to make a particular scan
terminate
  unsigned int scan_end_arg;   // parameters that influence this
termination

  unsigned int stop_src;       // what make the acquisition
terminate
  unsigned int stop_arg;       // parameters that influence this
termination

  unsigned int *chanlist;      // pointer to list of channels to
be sampled
  unsigned int chanlist_len;   // number of channels to be sampled

  sampl_t *data;               // address of buffer
  unsigned int data_len;       // number of samples to acquire
};
```

Las tarjetas USBDUX pueden configurase para realizar estas secuencias de muestreo o emisión de señal. El protocolo USB en modo de transferencia isócrono habilita unas tuberías que permiten la transferencia en frames o microframes que garantizan el volcado rápido de datos hasta el PC.

Por medio de las banderas de final de adquisición End Of Scan podemos reconocer la llegada asíncrona de una secuencia de muestras obtenidas por los canales habilitados. Fijando la tarjeta a una frecuencia de milisegundos, y gracias a la configuración del modo isócrono de transferencia en USB se habilita un sistema de temporización que permite la posibilidad de comportamiento determinístico del sistema, al menos en media.

En Comedi encontramos funciones que trabajan bajo sistemas en tiempo real, rt_comedi_command_data. Con estas funciones podemos realizar un sistema de adquisición/emisión de una muestra controlada. Los comandos de Comedi programan la tarjeta DAQ para una adquisición asíncrona de muestras de señal analógica. Las funciones rt_comedi_command_data_read/write permiten la adquisición de una única muestra de señal obtenida al final de una secuencia de adquisición/emisión. Los relojes y sistemas de temporización de la tarjeta DAQ ofrecen una buena precisión temporal, suficiente para el manejo de señales en el rango de milisegundos.

En el caso de la configuración de comandos para la adquisición de señal analógica, existe la posibilidad de realizar un registro que por medio de una bandera, la cual especifica el final de una secuencia de adquisición, actúa como un callback a través de un semáforo, es decir, una vez que la tarjeta DAQ en una configuración asíncrona a través del protocolo USB en modo de transferencia isócrono ha realizado la adquisición de una muestra, esta función crea un semáforo que regula e indica a Comedi la llegada de una muestra. Esta función rt_comedi_regiter_callback es necesaria para obtener sincronismo sobre el sistema asíncrono configurado. La tarea de adquisición queda bloqueada por el semáforo a la espera de más muestras. Según el fabricante el protocolo USB y por tanto la DAQ en

tiempo real funciona como máximo a 1 khz. En una configuración de la tarjeta a 1 kHz, y gracias a la garantías del modo isócrono, el desbloque del semáforo ocurre cada 1 ms. (véase www.rts.uni-hannover.de/rtai/lxr/source/addons/comedi/README?a=i386;v=3.3)

En el caso de emisión de señal para el control de sistemas, las señales a emitir son enviadas por el ordenador PC a la tarjeta. La tarjeta DAQ actúa como receptora de datos a una frecuencia fijada gracias a la configuración de los comandos. Los datos a emitir deben ser entregados con una frecuencia similar a la de la emisión de los mismos.

3.2.3 ComediRecord

Comedirecord es un programa osciloscopio que permite el manejo de la tarjeta USBDUX de una manera gráfica y visual ya que permite la proyección de los datos en tiempo real realizando una adquisición asíncrona y la selección de los canales por medio de una interfaz muy intuitiva. El programa a su vez genera archivos en formatos ASCII con extensión .dat compatibles con Octave o GNUPLOT. Este programa ofrece un filtro que elimina la corriente continua y un filtro Notch que elimina el ruido eléctrico que se genera por la alimentación de corriente en la frecuencias de 50/60 Hz. La frecuencia de muestreo o tasa de adquisición es fijada por parámetro aunque no ofrece el máximo potencial de las tarjetas DAQ USBDUX. La tarea de adquisición de datos condiciona la tarea de refresco de la imagen que muestra el programa osciloscopio: a tasas de muestreo de varias decenas de kilohercios, el programa no puede procesar los datos.

El funcionamiento de este programa es un ejemplo de aplicaciones open loop, la tarjeta DAQ USBDUX se configura para realizar una adquisición asíncrona sin condición de parada, por lo que la tarjeta DAQ de manera independiente realiza secuencias de scan periódicas de adquisición de muestras analógicas. Estas muestras son empaquetadas en tramas USB y enviadas en modo isócrono al PC host, el cual en la aplicación tiene una serie de buffers que albergan los datos muestreados que se van utilizando a la frecuencia que se necesitan.

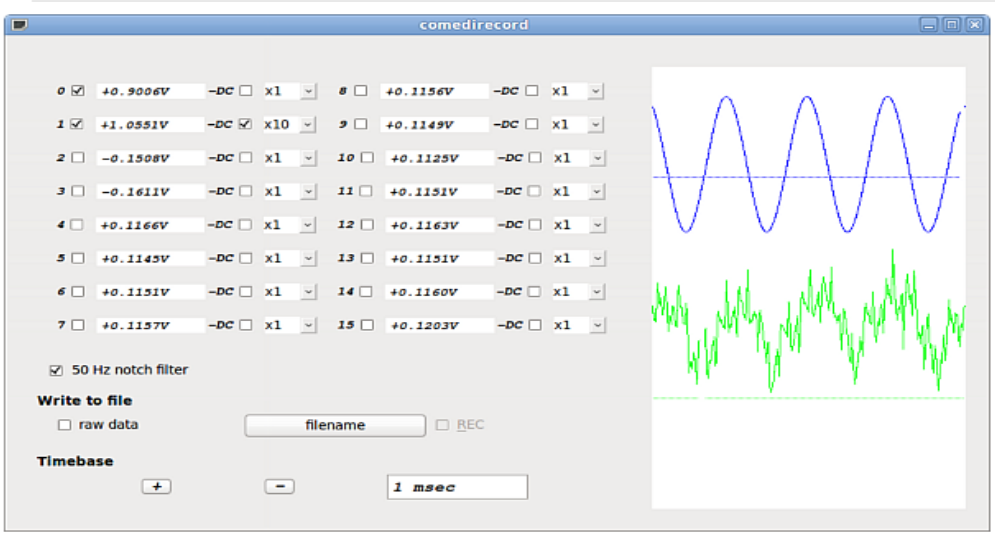


Figura 11 : Captura de pantalla con el funcionamiento de Comedirecord.

3.2.4 RTOS Sistemas operativos en tiempo real

Se desea diseñar un sistema que permita el manejo estricto temporal de la tarjeta DAQ para poder así realizar un sistema en tiempo real que monitoriza el funcionamiento de otro sistema. Por todo lo comentado en el apartado RTAI, y como solución intermedia, es elegido como sistema operativo un kernel genérico 2.6 con un patch RTAI compatible con este kernel, en concreto la versión 3.6. RTAI

RTAI es un patch de LINUX que consiste en una modificación del código fuente del kernel genérico LINUX. Este patch permite aumentar la funcionalidad del kernel genérico y le otorga las características de un sistema operativo en tiempo real. Este aumento de funcionalidad se da gracias a la instalación del Real Time Hardware abstraction law RTHAL. Este RTHAL consigue unir en una estructura simple todos los punteros a los diferentes componentes que manejan el kernel y sus funciones definidas. Las funcionalidad del kernel incluye las estructuras de datos y las funciones necesarias para manipularlos, las banderas de control del sistema, interrupciones externas o controladores de interrupciones programables, temporizadores, system call… A su vez respecta todas las funciones, estructuras de datos y marcos de LINUX para poder inicializar el RTHAL para el uso normal de LINUX (Dozio and Mantegazza, 2003). Por lo tanto, podemos resumir la funcionalidad del RTHAL en la captura y el manejo de las interrupciones software y hardware del sistema, todo el manejo de colas de tareas a la espera de ejecución, interrupciones hardware de periféricos, etc. y a su vez permitir la planificación de las tareas RT o hilos/procesos Linux concurrentemente (Muñiz, 2005).

El kernel Linux toma el control del componente hardware que RTHAL indique. El kernel de Linux puede en tiempo de ejecución, es decir una vez inicializado el PC, añadir funcionalidad por medio de la inserción y enlace de objetos código fuente, llamados módulos.

Otra gran ventaja de utilizar RTAI es que se tiene acceso tanto a las funciones a las que tiene acceso el kernel como a las funciones a las que tiene acceso el usuario manteniéndose un gran sincronismo y comunicación permitiendo así una ejecución simétrica de tareas tanto en espacio kernel como en espacio de usuario. Por lo tanto, se pueden desarrollar aplicaciones de tiempo real en espacio usuario sobre hilos estándar del kernel LINUX o por medio de tareas de RTAI. Esto se debe a que se tiene un manejo total de las interrupciones tanto en espacio kernel como en espacio usuario.

RTAI nos ofrece tres planificadores de procesos. Estos planificadores como se discutía anteriormente son totalmente *preempted* y las tareas que ejecuta el procesador pueden ser introducidas directamente para su ejecución sin tener que hacer uso de las interrupciones, por lo que LINUX no puede retrasar a la tarea en tiempo real. Los planificadores que, ofrece son: una versión para Uniprocesadores y SMP (symetric multi uniprocessor) y MUP (multi uniprocessor) para aplicaciones que se realicen de forma simétrica en uno o varios procesadores. En el caso SMP cualquier tarea que esté preparada para ser ejecutada (ready pipe) puede acceder a cualquiera de las procesadores libres o incluso adjudicarse tiempo de ejecución por lo que mantiene un compromiso de eficiencia y flexibilidad. En MUP cada tarea tiene asignada una CPU en concreto. RTAI especifica que las tareas kernel pueden ser movidas en tiempo de ejecución entre diferentes CPUs. No ocurre mismo con tareas LINUX e hilos de kernel (Dozio and Mantegazza, 2003).

Las políticas de planificación de procesos que los planificadores RTAI ofrece son:

- Preemptable First in first out,
- Round Robin
- Earling deadline first

Con estas políticas de planificación se construyen las arquitecturas sotfware para sistemas en tiempo real.

Existen dos principales arquitecturas para sistemas en tiempo real. Si la ejecución de las tareas se realiza de manera secuencial, las tareas se ejecutan de una manera fijada antes de iniciarse el sistema, el sincronismo y los recursos que necesitan compartir entre las tareas se conocen y la planificación se considera realizada offline. La ejecución de las tareas por lo tanto se realiza en orden y constituyen un ciclo ejecutivo. La otra arquitectura software consiste en una ejecución concurrente en el que las tareas compiten por los slot de tiempo que ofrece el procesador para poder ejecutarse. El orden de ejecución en este caso se puede asignar de manera estática, con un análisis antes de ejecución o de manera dinámica, mediante el análisis durante la ejecución que viene determinado por el nivel de prioridad que tenga la tarea.

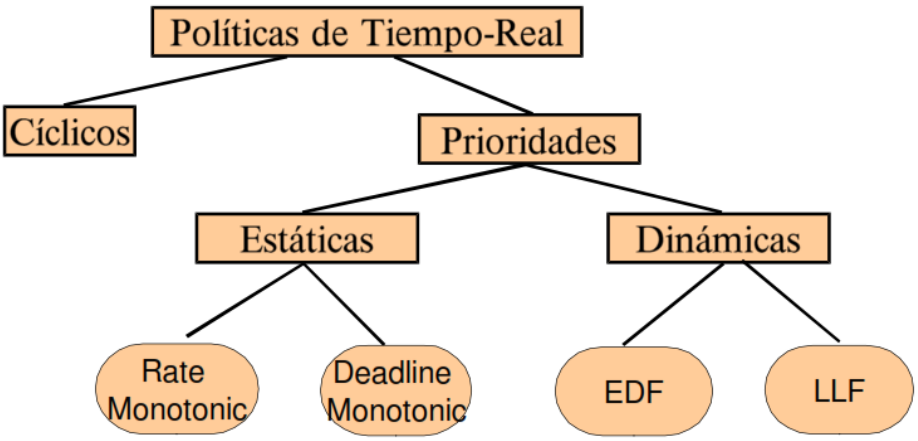


Figura 12: Planificaciones en los sistemas de tiempo real y algoritmo de planificación. (Ripoll Ripoll, 1997)

La ejecución concurrente es una forma de planificación que ofrece la posibilidad de introducir nuevas funcionalidades en tareas al sistema sin tener que redefinir toda la planificación, maneja bien las tareas esporádicas o aperiódicas, no es necesario dividir las tareas de duración larga, es el planificador el que realiza los cambios de contexto en el caso de que otra tarea necesite ejecutarse. Las desventajas de utilizar este tipo de planificación residen en la interactividad de sus tareas, la cual no está fija offline por lo que hay que implementar un mecanismo para el uso de recursos compartidos o utilización de mensajes entre las tareas. También la ejecución concurrente necesita un método de planificación, es decir necesita de un algoritmo de planificación y de un test de planificabilidad para poder asegurar el buen funcionamiento delo sistema. Estos algoritmos de planificación son más complejos que los firts come firts served o Round Robbin que utilizan los sistemas operativos tradicionales, en los que incluso las tareas no son expulsadas por el planificador,

sino que la tarea toma el control del procesador hasta completar su funcionalidad. Los algoritmos de planificación de estos sistemas están basados en prioridades que se asignan a las tareas y los planificadores son preemted o concurrentes por lo que pueden impedir o interrumpir la ejecución de una tarea si una tarea de mayor prioridad irrumpe en el sistema.

En la ejecución cíclica por el contrario no existen problemas de sincronismo ni de interactividad entre las tareas, no se puede dar un conflicto con el uso de un recurso por lo que no son necesarios mecanismos de exclusión mutua ya que los momentos de ejecución de las tareas se realizan de manera secuencial y se repiten de manera periódica. En este tipo de planificación no son necesarios algoritmos de planificación ya que las tareas se repiten de forma idéntica periódicamente y el análisis de planificación se realiza de manera visual por medio de un esquema de las trazas de ejecución.

3.2.4.1.1 Resumen de las características principales de RTAI:

En esta sección se especifican de forma resumida las principales características de RTAI.

Manejo básico de las tareas:

- Manejo de los tiempos de ejecución de las tareas y la conversión de tiempos a *system tick.*
- Asignación dinámica de prioridad.
- Posibilidad de elección de la política del planificador.
- Bloqueo y desbloqueo en la planificación.
- Sistema de suspensión y resume de las tareas temporizado para evitar bloqueos mutuos, cambios de contexto al finalizar la ejecución de la tarea, *task yield,* maneja tiempos de suspensión absolutos y relativos.
- Control específico para la ejecución de tareas periódicas.

Manejo de la memoria:

- Espacio de memoria compartida entre tareas.
- Los datos y variables son compartidos entre las tareas en espacio usuario y kernel.
- Asignación dinámica de memoria, es capaz de darle un espacio de memoria dedicado a la tarea durante un periodo concreto de tiempo de manera dinámica en tiempo de ejecución.

Otros aspectos importantes a destacar:

- Existe un registro de los objetos de RTAI, de forma que pueden ser referenciados a través de sus identificadores. Esto es una solución muy cómoda a la hora de implementar alguna aplicación.
- Ofrece un soporte hard real time tanto para tareas propias de RTAI como para los hilos de kernel.
- Manejo de la interrupción en espacio usuario, así se puede en espacio usuario implementar las interrupciones o el driver para el manejo del hardware. Los temporizadores y reloj del sistema. El sistema puede elegir la resolución temporal.

- También RTAI nos ofrece semáforos, incluye varios tipos de semáforos con el algoritmo de PHP priority inheritance protocol que evita los problemas de inversión de la prioridad, variables condicionales como signal o wait similares a las funciones del API de LINUX, Mailbox y FIFO para comunicación entre tareas o procesos, bits de sincronización como banderas, y señales para multitarea.

RTAI tiene adaptado totalmente, tanto en espacio usuario como en espacio kernel, una gran variedad de drivers para el manejo de distintas tarjetas, esta colección de drivers pertenecen a COMEDI y se instalan y compilan junto a RTAI. Las tarjetas DAQ utilizadas emplean el protocolo USB para comunicarse con el PC. El controlador de las tarjetas utiliza las funciones de la pila USB del kernel Linux. RTAI no ofrece una pila kernel utilizable en aplicaciones que corran en modo hard real time. El protocolo USB no ofrece garantías temporales en todos sus modos. Hay un proyecto en estatus pre-alpha, alpha que impleneta una pila USB versión 2.0 y sus host controladores EHCI, UHCI and OHCI para RTAI y conseguir una modalidad de actuación en modo estricto temporal o hard real time. El proyecto se llama USB2.0 for realtime y la página web del proyecto, www.developer.berlios.de/projects/usb20rt/.

3.2.4.1.2 Módulos del kernel

RTAI está diseñada como una herramienta modular la cual puede interaccionar con el sistema operativo de propósito general añadiéndole nuevas funcionalidades de tiempo real por medio de la carga de módulos de manera dinámica. Un módulo no es más que un trozo de código que se le añade al código del sistema operativo y que otorga nuevas funcionalidades al mismo.

Un ejemplo es el módulo RTHAL, mediante el cual se consigue el total control para el manejo del hardware por parte de las tareas definidas en tiempo real, por lo que ganan la máxima prioridad y autoridad para adquirir el control del procesador. Es decir, ninguna tarea de mantenimiento del sistema operativo Linux va a tener más prioridad que cualquier tarea de tiempo-real definida, a no ser que se especifique lo contrario, por lo que las tareas de tiempo-real no podrán sufrir retrasos en sus tiempos de ejecución y por tanto se puede asegurar el buen funcionamiento temporal del sistema.

RTAI posee dos planificadores diferentes que se cargan como módulos. Estos son los encargados de asegurar que las tareas que se ejecuten en el procesador encuentren sus constantes temporales sin verse perturbados por el sistema operativo LINUX (Racciu and Mantegazza, 2006). Ambos planificadores funcionan ofreciendo un servicio simétrico en tiempo real tanto dentro del espacio usuario como en espacio kernel y a su vez entre ambos espacios. La principal diferencia entre ambos planificadores es el tipo de objetos kernel que pueden manipular.

Rtai_lxrt es un co-planificador simple de LINUX/GNU, el cual soporta hard o soft real time para todos los objetos del planificador LINUX como procesos, hilos e hilos de kernel (Kthreads). El rtai_ksched no solamente soporta los mismo objetos que lxrt, también es capaz de manejar tareas kernel o módulos propios de RTAI. El cambio de contexto en las tareas kernels RTAI se realizan de manera más rápida que los hilos de kernel LINUX. Es decir, en aplicaciones en espacio usuario ambos planificadores son iguales. Si se realiza una aplicación en la que se tenga que realizar una acción cooperativa entre varios procesos

o hilos LINUX, es preferible que se realicen los cambios de contexto dentro el entorno LINUX. Si la aplicación requiere ser desarrollada en espacio kernel, las tareas de RTAI kernel requieren menores tiempos para realizar cambios de contexto que los hilos kernel que nos ofrece LINUX (Racciu and Mantegazza, 2006).

Debido a que las tarjetas DAQ USBDUX esta diseñadas y controladas en Comedi y actúan bajo la pila USB de kernel de LINUX, se utilizará el módulo planificador rtai_lxrt. Otros módulos requeridos debido a que ofrecen funcionalidad de sincronización, comunicación , gestión, etc… dentro de RTAI y el path (ruta de sistema de archivos) de localización de los módulos son (Mantegazza, 2006):

```
/usr/realtime/modules/rtai_fifos.ko;
/usr/realtime/modules/rtai_sem.ko;
/usr/realtime/modules/rtai_mbx.ko;
/usr/realtime/modules/rtai_msg.ko;
/usr/realtime/modules/rtai_tbx.ko;
/usr/realtime/modules/rtai_bits.ko;
/usr/realtime/modules/rtai_mq.ko;
/usr/realtime/modules/rtai_shm.ko;
/usr/realtime/modules/rtai_comedi.ko;
```

3.2.5 GNUPLOT Y OCTAVE: representación y procesado de datos.

Para representar los datos se ha utilizado GNUPLOT, una herramienta que permite la especificación de opciones y el control por línea de comandos. Está diseñada para la representación gráfica de datos y series temporales (www.gnuplot.info/docs_4.6/gnuplot.pdf). Con esta herramienta se representarán tanto las muestras de las señales monitorizadas, como las gráficas de los tiempos de cada ejecución.

Para este proyecto se realiza las grabaciones de los experimentos por medio de la USBDUX-fast mientras que la USBDUX-sigma implementa ciclos cerrados o la estimulación/emisión de alta precisión temporal. El fabricante de las tarjetas DAQ USBDUX ofrece un código de prueba en el que se entrega una plantilla para gnuplot llamada test.plt. Esta plantilla se encuentra en el Anexo: C Plantilla GNUPLOT.

Gracias a las grabaciones obtenidas deducimos la latencia de la tarjeta DAQ en la adquisición simple de señal, la periodicidad del ciclo cerrado o el jitter del sistema de estimulación. Por medio de Octave, (www.gnu.org/software/octave/), se procesan los puntos recogidos contabilizando los puntos que hay entre cada emisión de estimulación. La estimulación se produce en un momento concreto que viene determinado por un estado de excitación detectado en un mínimo de la señal que representa la actividad del sistema periódico que se utiliza para este proyecto. Este sistema periódico es el generador de señales o la neurona artificial que mantiene modos de excitación, lo cuales poseen diferentes periodicidades.

Octave es un intérprete de alto nivel de programación de todo tipo de operaciones y funciones matemáticas en un lenguaje similar al utilizado en Matlab. Octave al igual que los demás componentes software, programas o drivers utilizados en este proyecto es de licencia open-source.

3.3 Estimulación o adquisición de alta precisión

La estimulación o control de alta precisión temporal requiere una emisión de señal controlada tanto en duración como en momento de actuación. En muchos sistemas y aplicaciones la precisión de actuación de una tarea representa un factor crítico y muy importante de diseñado ya que la eficiencia del sistema puede depender de ello. El manejo estricto de los tiempos de actuación de las tareas y el sincronismo con la actividad del sistema son características de los sistemas en tiempo real, la utilización de un sistema en tiempo real es condición necesaria para asegurar unas condiciones mínimas de control de tiempos. La estimulación de alta precisión debe ser implementada sobre un sistema en tiempo real que permita asegurar la estimulación en un rango de tiempo limitado, estudiado y conocido. La temporización del sistema se puede realizar de dos maneras: modo síncrono o modo asíncrono.

3.3.1 Modo síncrono: Temporización por el PC

Una primera aproximación a la adquisición/emisión de alta precisión es a través del control temporal ofrecido por RTAI en el PC host. La comunicación con la tarjeta se realiza a través de los comandos síncronos ofrecidos en Comedi en sus funciones de adquisición/emisión simple de señal. La tarea periódica de RTAI realiza peticiones a la tarjeta la cual responde en un tiempo fijo. Las instrucciones síncronas están implementadas como comandos de precisión.

La precisión en la duración del estímulo viene dada en el bloqueo de la tarea o proceso que realiza una petición sobre la tarjeta DAQ. Este tiempo se puede entender como el tiempo de latencia que necesita la tarjeta para realizar la adquisición o emisión de una simple muestra analógica. Bajo el control de un RTOS, la tarjeta USBDX-sigma y cualquier componente controlado por USB ofrece, dependiente de la configuración de velocidad del protocolo USB, latencias de 1 ms (frame caso full-speed) o de 500 us (4 microframes caso high-speed) en la ejecución de las instrucciones de actuación simple con la tarjeta. La precisión de nuestro sistema está condicionada por esa latencia fija. El nivel de precisión en la actuación viene determinado por muchos factores como el tipo de fenómeno que genere el momento de actuación, el tipo de planificación elegido para el sistema o el tipo de sincronismo alcanzado entre la tarjeta DAQ y el PC host.

Gracias a RTAI podemos lanzar tareas que se ejecutan con mayor prioridad que cualquier tarea de control del sistema operativo Linux. Gracias a las funciones de temporización y a la posibilidad de crear tareas con la máxima prioridad del sistema, se posibilita la creación de un sistema de ejecución concurrente en el que la tarea RTAI única compite por el control del procesador con las demás tareas de control Linux desde una posición de mayor prioridad. El PC bajo el sistema operativo RTAI garantiza que se intentara servir las demandas de la tarea RTAI. El control del sistema RTAI se realiza de una manera no estricta total debido a las características del protocolo USB por lo que la ejecución de la tarea RTAI se realiza en una configuración soft real time.

La política del planificador RTAI permite la planificación concurrente *First in first out* basada en prioridades estáticas (Rate Monotonic). En los sistemas con planificación de ejecución concurrente la funcionalidad de emisión/adquisición está implementada en un tipo de tarea concreto provocando que sea el algoritmo de planificación el que determine el momento de actuación. Si la tarea de adquisición/emisión esta implementada en una tarea

periódica la prioridad de la tarea determina su orden de activación y la precisión de actuación queda fijada por el periodo de la tarea encargada de la activación de la emisión/adquisición en cambio si se trata de tareas aperiódicas será el mecanismo del servidor aperiódico el que determine el momento de actuación. El problema de los servidores aperiódicos es que asignan a sus tareas menor prioridad que las tareas periódicas que maneja el planificador e incluso se realizan de forma periódica para limitar el tiempo máximo de cómputo que utilizan las tareas aperiódicas y así no influyan en los tiempos de ejecución de las demás tareas críticas y periódicas del sistema. Las funcionalidades de emisión/adquisición se implementan sobre tareas periódicas RTAI, es decir el fenómeno que genera el momento de actuación de la adquisición/emisión es la activación de la tarea periódica.

Con esta configuración de emisión/adquisición síncrona y gracias a la funcionalidad de tiempo real de RTAI, que otorga la capacidad de la gestión de la ejecución de tareas en función de sus prioridades, más las latencias fijas que ofrecen las funciones de simple adquisición/emisión analógica de Comedi y más las garantías de precisión temporal de RTAI que posibilitan la creación de tareas periódicas, se diseña un mecanismo de precisión temporal. Este mecanismo está definido tanto en duración (latencia mínima de las instrucciones simples de adquisición/emisión de muestras analógicas de Comedi, cuyo valor depende de la configuración de la velocidad del protocolo USB), como en precisión de actuación, determinada por la precisión en la ejecución de tareas por parte de RTAI.

En el caso de la emisión, la forma de determinar la fiabilidad del sistema adoptada en este proyecto se logra a través de la creación de microeventos. Un microevento es la emisión de señal periódica mínima posible ofrecida por el sistema, es un pulso de valor de tensión configurable activo un tiempo fijo que vuelve a valor cero al pasar el mismo tiempo. Por ejemplo, en una configuración full-speed en la cual la latencia de duración de emisión de señal analógica está fijada en 1 frame de USB o 1 milisegundo, el tiempo mínimo para la generación de un micro evento son 2 milisegundos de latencia mínima, por lo que se puede realizar una tarea de ejecución periódica cada 2 milisegundos. El análisis de emisión continua de microeventos será el mecanismo elegido para determinar la viabilidad del sistema.

En el caso de la adquisición, la periodicidad de la tarea determina la tasa de muestreo o la precisión temporal del sistema. La instrucción de lectura síncrona ofrecida en Comedi y para a tarjeta USBDUX-sigma está configurada para ofrecer latencias de 2 ms.

Se analiza la eficacia de la estimulación de alta precisión integrada en un sistema con planificación de ejecución concurrente gobernado por la prioridad de la tarea. Para ello se crea un programa que ejecuta, en una planificación FIFO concurrente, una tarea RTAI de prioridad máxima periódica que simula una ejecución cíclica de emisión de pulsos, en la que se realiza una activación de la tarea en momentos temporales conocidos y las latencias de las instrucciones síncronas de simple adquisición o emisión fijadas por la duración de la trama USB configurada entre el PC y la tarjeta DAQ. El sistema debe ser considerado un sistema soft real time, en el que la pérdida de un periodo o de una muestra no suponen el colapso del sistema y las latencias están dadas en media. La configuración de la tarjeta USBDUX-sigma se realiza a través del protocolo USB en un modo de transferencia en bulk, este modo de configuración no es adecuado para sistemas en tiempo real estrictos debido a que no tiene garantías de envió y realizan control de errores. Del mismo modo se realiza un programa para la adquisición síncrona de señal analógica.

3.3.2 Modo asíncrono: Temporización de la DAQ

En este modo de temporización del sistema las restricciones temporales no están determinadas por el PC sino por los relojes internos de la tarjeta DAQ. La tarjeta posee un modo de transferencia de datos en USB isócrono que permite el volcado o envió de datos entre el PC y la DAQ con unas garantías temporales, en una latencia conocida. La temporización del PC no es crítica y puede desarrollarse sobre sistemas de propósito general. Incorpora sincronismo y posibilidad de adquisición/emisión de una muestra sobre este sistema asíncrono precisa de un alto grado de control temporal, por medio de RTAI y las librerías de Comedi para tiempo real, kcomedilib, se construye un sistema síncrono para manipulación de señal analógica. En sistemas de propósito general y en espacio usuario Comedi ofrece instrucciones para el manejo de datos adquiridos de forma asíncrona, como comedi_poll. Otra manera de realizar la lectura de datos o escritura a través de las funciones estándar read y write ofrecidas por Linux. La lectura por medio de read y gracias al sistema de archivos de Linux donde todo tiene su archivo de texto, se realiza lecturas de un número determinado de bytes que se guardan en un buffer.

En este modo asíncrono y bajo Comedi ofrecido para RTAI y las condiciones temporales están impuestas por la tarjeta DAQ USBDUX, se obtiene sincronismo por medio de activación de banderas o mecanismos de control que comuniquen la llegada de nuevos datos, las garantías de envió en un tiempo máximo que ofrece el modo USB isócrono y las funciones de adquisición de una muestra obtenida a través de la configuración de un comando se construye un sistema síncrono de adquisición de señal analógico.
Los comandos permiten que se exploten todas las características de la tarjeta DAQ, la precisión está garantizada. Los tiempos de duración de los microevenos o pulso mínimo periódico están condicionadas por la configuración de la tarjeta DAQ, en concreto por el argumento de inicio de secuencia o scan (scan_begin) que se configura para funcionar respecto a una señal de reloj precisa a una frecuencia fijada por parámetro también.
El sincronismo se obtiene gracias a la librería de Kcomedilib la cual ofrece la función rt_comedi_register_callback, esta función habilita un semáforo para Comedi y así conocer el momento de llegada de muestra al PC. El protocolo USB en modo high-speed asegura latencias fijas de 125 micosegundos. Por medio de rt_comedi_command_data_read se obtiene una muestra. La llamada a esta función se realiza cada vez que el semáforo habilitado por medio rt_comedi_register_callback y que contrala el final de una secuencia de adquisición se active. En la documentación ofrecida por RTAI acerca de Comedi, se explica con detalle el uso de las funciones de Comedi que posibilitan la construcción de un sistema síncrono sobre un sistema asíncrono construido a través de la configuración de comandos que realizan secuencias periódicas de adquisición de muestras analógicas.

El caso de la escritura asíncrona posibilita un sistema de estimulación de precisión gestionada por la DAQ. La precisión en la duración está asegurada gracias a la tarjeta DAQ, en el caso de comunicación entre el PC y la DAQ síncrono la duración estaba condicionada por la configuración de la velocidad el protocolo USB, en el modo asíncrono es la tarjeta la que por medio de su temporización interna se configura a una frecuencia de muestreo y emite pulsos de duración precisa a esa tasa. Los datos a enviar deben estar disponibles en la tarjeta DAQ a un ritmo similar al que son emitidos. Las condiciones temporales sobre las funciones de emisión son menores que en el caso síncrono, siempre que el dato este antes de que la señal de reloj interno determine un nuevo momento de emisión, la emisión de señal analógica se reduce con un alto grado de precisión, acorde con la características de la tarjeta que ofrecen tasas de muestreo de milisegundos.

3.4 Ciclos cerrados de estimulación gobernados por objetivo

Como se describía anteriormente los ciclos cerrados de estimulación gobernados por objetivo o dependientes de la actividad deben ser realizados bajo un sincronismo máximo que se pueda con el sistema sobre el que se realice la experimentación. Por ello es necesario un control estricto del tiempo por lo que RTAI es totalmente necesario para el desarrollo de estos ciclos. Los sistemas sobre los que se realiza la experimentación, por ejemplo la neurona artificial, son monitorizado y estimulados por medio de las funciones de adquisición o emisión de señal analógica que se encuentran en el proyecto Comedi controlados por una planificación de ejecución cíclica implementada en tareas en espacio usuario en un PC con sistema operativo Linux con patch RTAI compilado. La tarjeta DAQ USBDUX-sigma es la encargada de la ejecución de las instrucciones de adquisición/emisión. La implementación de estos ciclos sirve para evaluar la precisión de la adquisición/emisión de señal.

Al igual que en la adquisición/emisión de alta precisión el sistema se puede realizar de dos formas, por medio de las funciones síncronas y dejando al PC la imposición de las restricciones temporales del sistema y el uso de las tarjetas se entiende como el uso de un recurso o acceso a zona critica o de forma asíncrona a través de la configuración de la tarjeta DAQ y la realización de un control, que dota de sincronismo al sistema. Las funciones de Kcomedilib posibilitan la manipulación de una muestra simple sobre una tarjeta configurada para realizar secuencias periódicos de emisión/adquisición de muestras a tasa fija configurada en el comando de Comedi.

Para una configuración síncrona del sistema bajo la versión 2.0 de USB, la adquisición y emisión de señal analógica por medio de la DAQ USBDUX-sigma se realiza por medio de la ejecución de instrucciones de latencia fija emitidas de manera síncrona desde el PC con RTAI. La tarjeta DAQ realiza su funcionalidad de adquisición o emisión bloqueándose durante un 1 milisegundo de tiempo de latencia o frame USB en versión USB 2.0 fast speed(1 kHz) o en high-speed a 0.5 milisegundos.

En modo asíncrono la tarjeta impone las condiciones temporales de adquisición y emisión y es configurada para funcionar a una tasa fija. Los datos se envía por el PC a través de USB a la máxima velocidad permitida suficiente como para dotar de espacios temporales o ventanas de emisión a los datos que la DAQ USBDUX-sigma necesita adquirir/emitir (high-speed, 125 microsegundos o microframe USB). Habilitado el canal de comunicación PC a DAQ para funcionar correctamente, se realiza la adquisición/emisión de una muestra concreta de manera controlada gracias a las funciones de Kcomelib que permiten la emisión/recepción de una única muestra.

El ciclo al completo puede realizarse en una misma tarea RTAI de máxima prioridad sobre un sistema con planificación FIFO concurrente que se ejecute de manera periódica. La ejecución concurrente ofrecida por RTAI posibilita la creación de los ciclos cerrados sobre hilos que realizan tres tareas independientes, monitorización, procesado de datos y decisión de estimulación y estimulación. Esta implementación de los ciclos por medio de áreas que se ejecutaran de manera concurrente genera la necesidad de incorporar funciones de sincronismo que ordenen las diferentes tareas que componen el sistema complicando la implementación del mismo, por otra parte permite mayor flexibilidad al sistema. Se decide utilizar una única tarea RTAI en las que se desarrollen las tres tareas que componen el

ciclo. Estas tareas por tanto poseen un bucle infinito o de duración deseada en el que se realiza la adquisición de una muestra de señal, se realiza un procesado de los datos adquiridos para determinar si se necesita estimular o emitir señal debido a que hemos detectado un objetivo y estimulación si procede. La precisión del ciclo cerrado de estimulación dependiente de la actividad estaría sujeta a la periodicidad. El periodo mínimo se fija en función del peor tiempo de ejecución del bloque que ocurriría en el supuesto positivo de tener que estimular. La adquisición se realizaría con una tasa de muestreo igual que la periodicidad de un ciclo.

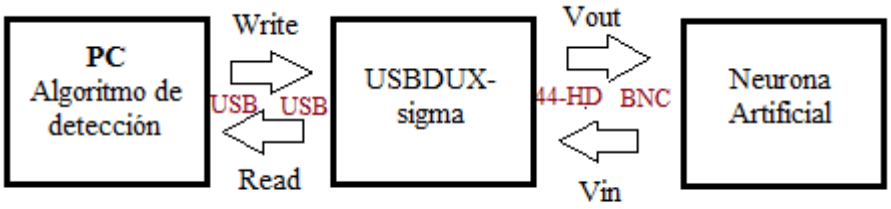


Figura 13: Esquema del sistema de tiempo real para la elaboración de ciclos cerrados dependientes de la actividad gobernados por objetivo. Especificado un objetivo, se adquiere la señal a través de la DAQ USBDUX-sigma (Vin) y es enviada al PC por USB (read), en el algoritmo se detectan los eventos relevantes, se actualizan los parámetros de la representación interna de la interacción, se ajusta la estimulación o el explorador de estímulos y se envían las señales de control o muestras a la tarjeta DAQ USBDUX-sigma (write) para que emita la señal analógica sobre la neurona artificial (Vout).

La planificación de las tareas se realiza por medio de un análisis de la traza de ejecución, tanto si se realiza en modo asíncrono como síncrono. En el caso síncrono, las instrucciones tienen unas latencias fijas, y en el caso asíncrono el programa que manipulen las tarjetas es bloqueado un tiempo fijo a la espera de la llegada de un nuevo dato obtenido con precisión temporal en la tarjeta DAQ USBDUX. También se utilizan las funciones de espera y temporización ofrecidas en RTAI. Aprovechando la potencia y precisión de los relojes de las DAQ y la velocidad de transmisión del protocolo USB 2.0 en high-speed se elabora un sistema soft real time que realiza ciclos cerrados de estimulación dependientes de la actividad.

4 Desarrollo

En el apartado diseño se analizan los problemas y se proponen las formas más correctas para conseguir implementar el sistema propuesto en el proyecto. En esta sección se explican los detalles de desarrollo de los diferentes componentes hardware o software que se elaboran en este proyecto para conseguir implementar un sistema en tiempo real soft que permita el manejo de las tarjetas DAQ USBDUX a través de USB de manera precisa creándose el soporte necesario para la realización de estimulación de alta precisión temporal (milisegundos de precisión) que se probará en los ciclos cerrados de estimulación de alta precisión dependientes de la actividad o gobernados por objetivo.

4.1 Hardware

Como se explicaba en la fase de diseño y por las características que ofrece cada tarjeta DAQ se opta por el uso de la USBDUX-sigma como DAQ encargada de la realización de la adquisición/emisión de alta precisión. Una vez demostrada la capacidad y límites de la estimulación/adquisición de señal analógica se analizara la realización de los ciclos cerrados de estimulación gobernados por objetivo. La USBDUX-fast será utilizada para el registro de señal en modo asíncrono de adquisición. Cada tarjeta se utiliza para un cometido diferente y su nivel de sincronismo entre la tarjeta y el PC tiene características temporales distintas.

Se decide utilizar dos PC, el primero para la implementación del sistema en tiempo real gracias a la instalación de RTAI, el cual maneja la adquisición o estimulación de milisegundos de precisión implementada sobre un mecanismo síncrono ofrecido por Comedi y la temporización de RTAI o sobre una configuración asíncrona donde se obtiene un alto grado de precisión gracias a la temporización de la tarjeta y se obtiene sincronismo por medio de las funciones de adquisición de una muestra en un comando ofrecido en las librerías Comedi para tiempo real, Kcomedilib. El otro PC sin necesidad de características temporales estrictas, actuará bajo el control de un sistema operativo de propósito general Ubuntu 12.04, ya que actuará la tarjeta DAQ de manera asíncrona, la temporización de adquisición la impone la DAQ. Este PC junto con la DAQ USBDUX-fast será utilizado para realizar una grabación del sistema completo, tanto sistema de emisión como sistema del estudio, que es en una primera fase un generador de onda sinusoidal y después es la neurona artificial.

Resumen del hardware utilizado:

- PC Pentium 4 sistema operativo Ubuntu con kernel 2.6.38 (UBUNTU 10.04)en el que se realiza un patch RTAI que maneja por medio de los drivers módulos y funciones de Comedi la DAQ USBDUX-sigma
- PC laptop core i7 sistema operativo de propósito general UBUNTU 12.02 que maneja por medio de Comedi la DAQ USDDUX-fast

Los sistemas en los que se testa la eficacia en de las tarjetas DAQ son:

- Generador de señales: el generador de funciones modelo GF1000 de la compañía PROMAX que emite señales de tipo sinusoidal, pulsos cuadrados o señales triangulares. La configuración de la frecuencia de señal se fija por medio de una

rueda que permite fijar valores del 1 al 10, los órdenes de magnitud de la frecuencia se aplican por medio de botones. La precisión de este generador de señal no es muy buena pero suficiente para realizar pruebas de detección de mínimos o máximos, que representan momentos necesarios de estimulación.

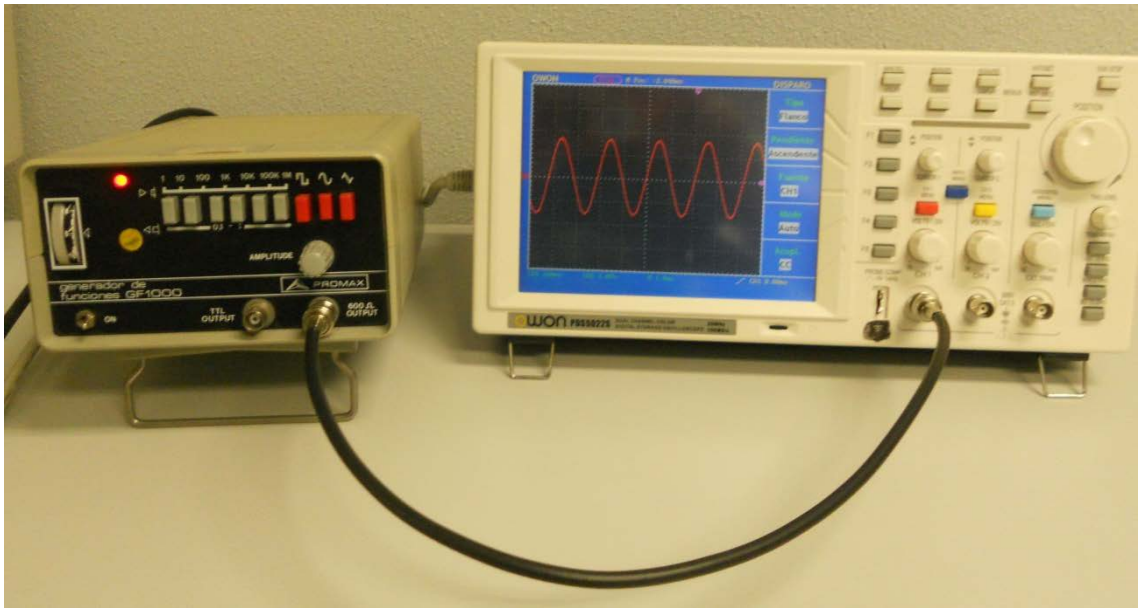


Figura 14: Foto Osciloscopio PDS5022S y generador de funciones GF1000

- Neurona artificial: es la neurona desarrollada en el GNB explicado en el apartado de diseño, llamado neurona artificial. La neurona en un primer momento no tenía la salida bufferizada por lo que la impedancia de salida provocaba errores en el registro de señal. Se coloca un buffer o seguidor de tensión a la salida para simular una impedancia casi nula y que no se provoque perdida de potencial por la corriente que se deriva.

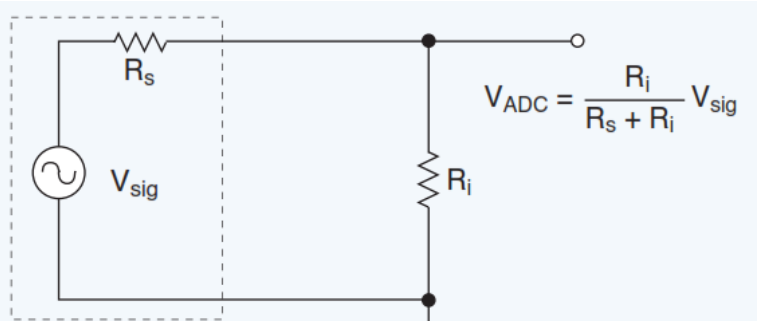


Figura 15: Efecto que se produce cuando la salida de la nuerona no estaba bufferizada.
La resistencia interna de la neurona antes de colocar el buffer es R_s, con el buffer casi nula.(Computing)

Osciloscopio: Durante todo el desarrollo y la integración de las pruebas se cuenta con un osciloscopio modelo PDS5022S de la compañía OWON que es utilizado para medir y controlar el nivel de emisión de señal que se produce desde cualquier elemento del sistema para evitar que se provoquen daños en la circuitería del componente receptor. La adaptación de señales entre cada componente del sistema es necesaria.

4.1.1 Conexiones

Siguiendo los principios de diseño, aislamiento y protección contra el ruido o fenómenos electromagnéticos, flexibilidad, fiabilidad y estabilidad mecánica se construye una conexión para cada tarjeta de adquisición USBDUX.

La conexión de los canales analógicos correspondiente a los pines que son elegidos para la implementación del sistema, se realiza por medio de unos cables de cobre de 11mm que se sueldan al conector. Estos cables son conectados a los cables de micrófono. A cada cable que se suelda, se le da un baño de estaño en la punta para favorecer la soldadura al conector de 44 pins. Es importante realizar la soldadura a gran temperatura, unos 400ºC, ya que no hay problema de que los componentes a soldar se quemen, y así se evita efectos de soldadura fría y por tanto que no se realice la conexión correctamente. Las conexiones de los canales y sus cables son aislados por medio de termoplax, el termoplax es un material plástico que se deforma y amolda al aplicarle calor.

Otro aspecto importante es cómo se lleva a cabo la soldadura de todos los pines de tierra a un punto. La soldadura de las tierras se realiza a través de un cable conectado en trasversal, este cable es de 7 mm y se le da un baño de estaño que une al mismo punto físico todas los pines de ground y los canales que no serán utilizados, realizando la conexión de esta forma combatimos los fenómenos de crosstalk entre los diferentes canales de lectura/escritura y se reduce el ruido.

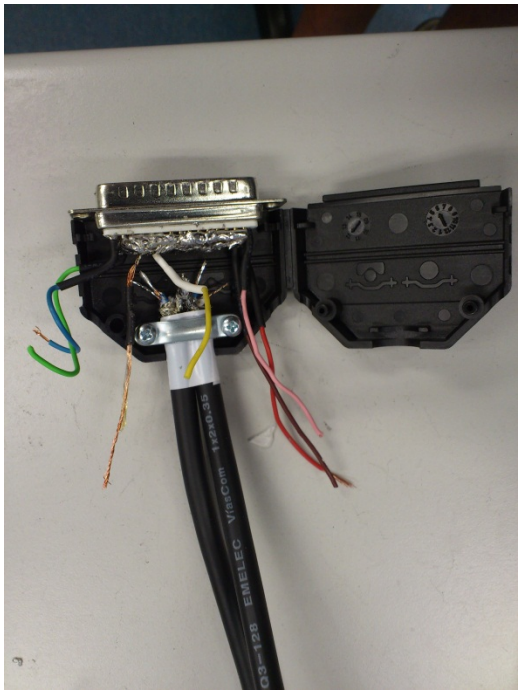


Figura 16: Elaboración de las conexiónes del conector de 44 pines HD para las DAQ´s USBDUX.

Las tierras deben formar un circuito cerrado para no producir interferencias sobre el sistema. Los canales deben cerrarse en el caso de no ser utilizados. Para darle mayor estabilidad mecánica una vez realizada la conexión de los pines del conector 44 pin HD se monta todo sobre una carcasa protectora la cual es inunda de silicona líquida. Con la silicona se ayuda al aislamiento y a su vez se le quita tensión mecánica a la soldadura.

4.1.1.1 Conexiones para la USBDUX-fast

La conexión de la USBDUX-fast se construye con dos canales analógicos de entrada. Para posible futura utilización se habilitan conexiones al pin de alimentación y al pin para la posible temporización de la tarjeta de manera externa. La longitud del cable es de 1.5 m.

Figura 17: Conexión para la DAQ USBDUX-fast. Se realiza la conexión para dos canales analógicos de entrada.

Es empleado un único cable de micrófono, conectado en un extremo al conector 44 pin HD y por otro lado finalizado en conector BNC y en minibanana

Son realizadas las siguientes conexiones:

PIN	Función	Conector
15	Canal 0 entrada	BNC
16	Canal 16 entrada	Mini-banana
32	Alimentación +5 V	No realizado
31	Temporización DAQ	No realizado
Demás	Tierra	Mini-banana

Tabla 6 Pines y conexiones realizados para la USBDUX-fast.

Los demás pines son conectados a tierra en el propio conector de 44 pin HD. Los canales que no se utilizan no deben quedar en circuito abierto. En el caso del cable con los canales analógicos acabados en conectores BNC se utilizan terminaciones para conectores BNC de la misma impedancia que los conectores para evitar fenómenos de reflexión de onda.

Para poder llevar a tierra el canal 16 que termina con un conector mini banana, se realiza una conexión de un cable de 11 mm con la maya que recubre los canales analógicos, Así se obtiene una conexión a tierra y así poder deshabilitar el canal analógico que no se utilizan en ese instante para la lectura de señales. A su vez se realiza una conexión auxiliar a tierra utilizada para la conexión de nuevas etapas de adaptación de señal.

4.1.1.2 Cable para la USBDUX-sigma

La conexión para la USBDUX-sigma es diseñada y construida con cuatro canales analógicos, dos de entrada y dos de salida. A su vez posee canales de alimentación por si es necesario escalar el sistema e introducir una etapa que necesite de alimentación.

Figura 18: Conexión para la DAQ USBDUX-sigma. Se realiza la conexión para un canal analógico de entrada y otro de salida.

Los cables escogidos en diseño poseen dos núcleos ambos aislados por materiales plásticos entre ellos y a su vez aislados del exterior por un mayado que se conecta a la tierra del sistema, en cada cable hay un canal de entrada y otro de salida. Estos cables son conectados en un extremo al conector de 44 pin HD y por el otro extremo es crimpado a conectores BNC y conectado a conectores banana.

Son realizadas las siguientes conexiones:

PIN	Función	Conector
41	Canal 0 salida unipol	BNC
42	Canal 1 salida unipol	No realizdo
16	Canal 0 entrada	BNC
15	Canal 16 entrada	N realizado
32	Alimentación +5 V	No realizado
31	Alimentación -5 V	No realizado

Tabla 7: Pines y conexiones realizados para la USBDUX-sigma.

Los demás pines son conectados a tierra en el propio conector de 44 pin HD.Los canales que no se utilizan no deben quedar en circuito abierto. En el caso del cable con los canales analógicos acabados en conectores BNC se utilizan terminaciones para conectores BNC de la misma impedancia que los conectores. Las conexiones no realizadas también se conectan a tierra por medio de un punto de soldadura.

4.1.2 Adaptación de señal

Circuitos de atenuación de gran señal gran señal. Se realiza un divisor de tensión para poder adquirir señales de valores comprendidos:

- [-7 – 7] V caso USBDUX-fast.
- [-13 - 13] V caso USBDUX-sigma.

Las resistencias elegidas son el modelo PR5Y 510R y PR5Y 200R de Farnel, pueden trabajar hasta potencias de 0.5 W y trabaja linealmente en un amplio rango de frecuencias. Han sido elegidas estas resistencias por su reducido valor de tolerancia de fabricación ±0.5% y reducido valor de coeficiente de temperatura del resistor, 15 partes por mil cada grado de temperatura Celsius que se modifique le temperatura de la resistencia. También las resistencias tienen un comportamiento lineal en los rangos de temperatura en los que el sistema será desplegado o utilizado. Así asegurarnos que los valores que las resistencias que toma el divisor de tensión no varían mucho y el error que nos genera en la cuantificación que se produce en la adquisición sean menor. En los diferentes programas de adquisición de datos o en los programas de procesado de datos obtenidos por la tarjeta hay que aplicar un escalado inverso al producido por el divisor de tensión. Esta fase de adaptación tiene como salidas conexiones realizadas en mini-banana. Se realiza adaptadores de mini-banana a BNC y viceversa.

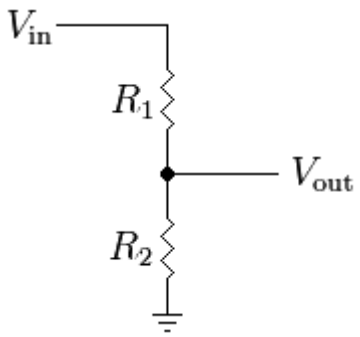


Figura 19: Circuito esquemático de divisor de tensión pasivo.
(Imagen de http://commons.wikimedia.org/wiki/File:Resistive_divider.png).

En nuestro caso las resistencias del circuito atenuador toman valor:

- R_1= 100 Ω (200||200 resistencias en paralelo)
- R_2=1040 Ω (520+520 resistencias en serie)

Con este circuito logramos que la señal se reduzca en amplitud pero no pierda ninguna característica de señal, frecuencia, fase. Cada señal de entrada será escalada por 0.096 aproximadamente 0.1.

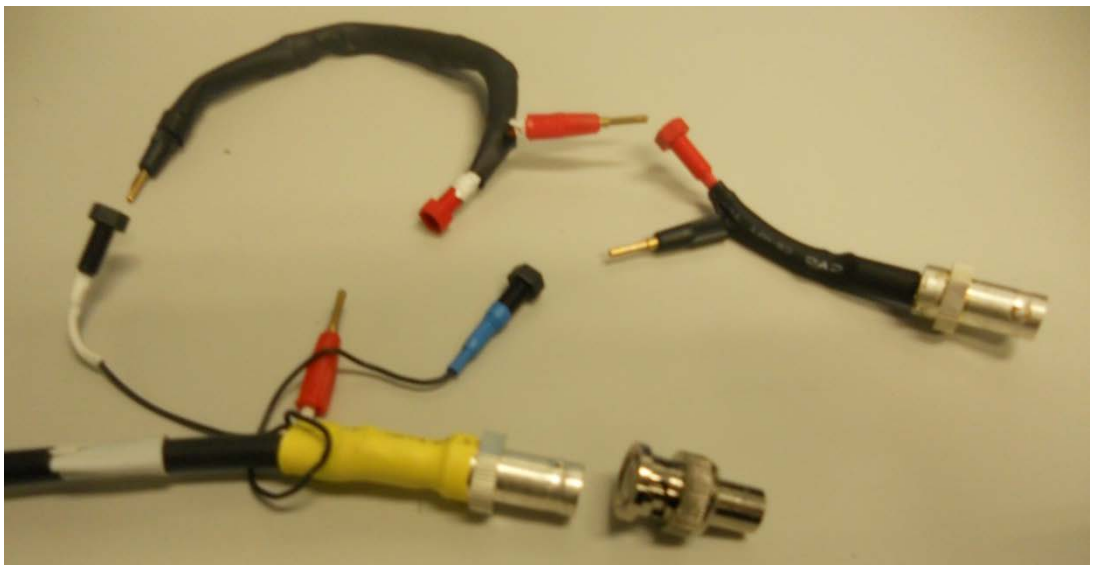


Figura 20: Circuito atenuador de aproximadamente 10 órdenes de magnitud, conversor mini-banana a BNC y terminación BNC para canales en desuso.

4.2 Software

4.2.1 Instalación de las herramientas del sistema

Para el desarrollo del proyecto es necesaria la instalación del sistema operativo RTAI en la máquina que controlara la estimulación y adquisición con restricciones temporales que se utiliza para controlar la ejecución del ciclo cerrado de estimulación dependiente de la actividad. Por otra parte se realza una partición e instalación de un sistema operativo de propósito general, en concreto, una distribución Linux Ubuntu 12.04, en la que se realizaran la grabación de lo que ocurra en el sistema, tanto para registrar la estimulación provocada desde el PC con Linux Ubuntu 10.04 y RTAI a través de la DAQ USBDUX-sigma como el registro de la señal que emite el sistema estimulado, la cual expresa el estado de actividad del sistema de estudio. Para este proyecto se utiliza un componente que simula el comportamiento de una neurona. Esta neurona artificial está fabricada en el laboratorio del GNB y junto al generador de señales serán los sistemas de experimentación que se utilicen.

En ambos sistemas se instala o se disponen de los módulos de Comedi. Las tarjetas DAQ USBDAX trabajan bajo el control de las librerías Comedi y de su módulo controlador específico para la tarjeta. El controlador o driver para la tarjeta DAQ USBDUX-sigma que se encuentra en la versión más moderna de Comedi está diseñado para funcionar en kernel superiores a la versión de kernel 2.6.38 de la distribución Ubuntu 10.04. Es posible descargar el módulo para la versión 10.04 de la página del fabricante.

4.2.1.1 Instalación de RTAI

La instalación de RTAI se realiza por medio de un patch que se compila junto a un kernel genérico 2.6.38. Este kernel genérico esta la distribución UBUNTU 10.04. El sistema operativo de UBUNTU está compuesto de su kernel llamado LINUX y otras aplicaciones de GNU. La instalación se realiza sobre un PC Pentium 4 a 3.2 GHz.

Los kernel genéricos están diseñados y formados para poder manejar el mayor número de máquinas o PC, no están diseñados para un Hardware específico. Con la compilación del Kernel genérico se consigue por una parte la optimización del hardware específico de la máquina y a su vez la eliminación de módulos que dan soporte a dispositivos que no están integrados en el sistema o no es adecuada su activación.

Los pasos de la instalación están detallados en el manual de instalación que se encuentra en el anexo. El manual instala también la extensión de Comedi para RTAI, librerías y módulo necesario para el desarrollo de un sistema en tiempo real (véase Anexo: A.Manual de instalación RTAI).

4.2.2 Inicialización del sistema

En la inicialización del sistema operativo UBUNTU se cargan los módulos necesarios para dar soporte a la funcionalidad básica que posee el sistema operativo, las tarjetas son plug and play por lo que son reconocidas de manera automática por el PC. Para poder desarrollar o implementar un sistema en tiempo real es necesario cargar los módulos de RTAI, los módulos de Comedi y los módulos driver de las tarjetas DAQ.

En el anexo podemos encontrar el script de shell el cual carga los módulos RTAI, Comedi y de la tarjeta DAQ USBDUX-sigma, así como realiza la configuración de las tarjeta DAQ (véase Anexo:B. Script de inicialización del sistema RTAI y de las USBDUX) .

4.2.2.1 Inicialización de RTAI

Como ya sabemos RTAI está diseñado de manera modular. Estos módulos se pueden cargar de manera dinámica una vez inicializado el sistema operativo. Pueden cargar de manera dinámica una vez inicializado el sistema operativo. Son instaladas las herramientas necesarias para la manipulación de los módulos. Es imprescindible ser root para poder usar estos comandos(Racciu and Mantegazza, 2006).

Las principales funciones pertenecen al modutils package:

- /sbin/insmod inserción en el sistema del módulo
- /sbin/lsmod especifica los módulos cargados en el sistema
- /sbin/rmmod descarga un módulo cargado en el sistema

Los módulos de RTAI pueden ser cargados por medio de modutils, como cualquier módulo del sistema operativo. Modprobe es otra herramienta de Linux que realiza la misma función que insmod pero de manera más genérica ofreciendo más funcionalidad que únicamente la inserción de módulos. Modprobe realiza búsqueda de dependencias del módulo insertado. Existe otra herramienta propia de RTAI, rtai_load, que permite cargar los módulos necesarios para realizar la funcionalidad necesaria. (http://www.rts.uni-hannover.de/rtai/lxr/source/doc/maintainers/runinfo.txt?v=3.3).
El fichero rtai_load carga a través de un fichero .runinfo los módulos que le indican. Este fichero permite la carga e inicialización de RTAI de manera automática, en estos fichero se describe las dependencias de los módulos así como las acciones de carga e inicialización o lanzar ejecutables necesarios para la aplicación. El formato del archivo es el siguiente:

Tarjet_name: modules dependences: run action: init_comment
Lxrtwrite: lxrt+fifos+comedi: lxrtwrite: control_c

Tarjet es el nombre simbólico que le damos al proceso que se ejecutará y a continuación se cargan los módulos y las dependencias a utilizar, seguido de dos puntos se colocan las diferentes acciones a realizar por la Shell, estas se ejecutarán de manera secuencial. Por último se puede colocar un mensaje que será proyectado en la Shell después de realizar la acción. El mensaje control_c, es un mensaje estándar que invita al usuario a tipear ctrl+c para descargar los módulos cargados para realizar la acción.

Se carga el módulo encargado de la planificación rtai_lxrt.ko, el módulo encargado de la memoria compartida rati_shm.ko.Es importante realizar la carga del módulo de Comedi compilado junto a RTAI, este módulo se encuentra en lib/modules/(uname -r)/comedi/comedi. También son necesarios los módulos kcomedilib, comedi_fc y rtai_comedi.

Realizando un lsmod comprobamos que todos los archivos están cargados.

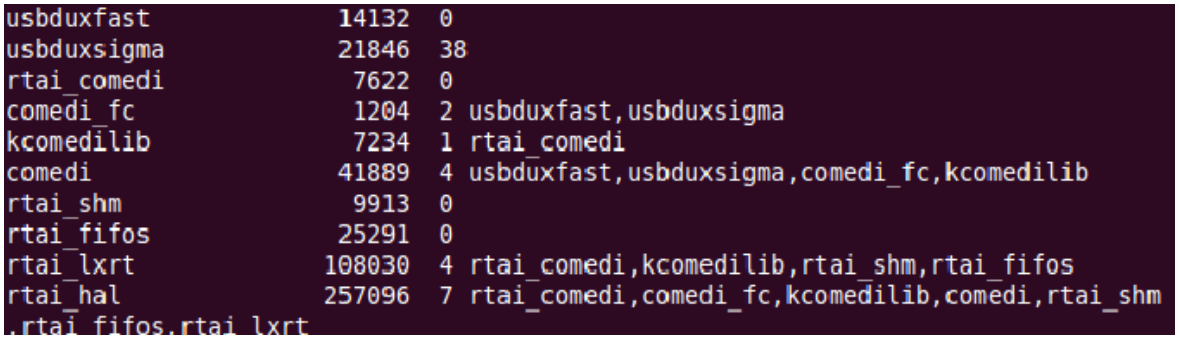


Figura 21: captura salida de lsmod. Observamos los módulos necesarios para el sistema.

4.2.2.2 Inicialización de las tarjetas DAQ USBDUX

Las tarjetas son ideadas como plug and play para todos los sistemas operativos de kernel superior a 10.04. La tarjeta DAQ USBDUX es conectada y reconocida por el PC, para comprobar que la tarjeta DAQ está reconocida por el sistema es conveniente revisar con dmesg.

Una vez que la tarjeta es conectada y reconocida es necesaria inicializar y configurar la tarjeta DAQ, se le asigna un fichero /dev/comedi y se le indica el driver módulo de COMEDI que se integrara en el kernel de manera dinámica y que maneja las tarjetas.
Comedi ofrece una herramienta de configuración, comedi_config.

En el caso de la USBDUX-sigma que será utilizada en un sistema con RTAI, es necesario cargar el modulo driver que controla la tarjeta. El modulo driver ofrecido en Comedi difiere del driver de la USBDUX-sigma para versiones de Linux 10.04, ofrecido por el fabricante en su página web. El nuevo módulo descargado y compilado se carga de manera manual en el sistema cada vez que inicializa el sistema.

Es aconsejable realizar un comedi_test para verificar que la tarjeta está correctamente configurada. Salidas de los comedi_test en el Anexo véase Anexo: Manual del programador: 1.3. Comedi_test para la USBDUX-fast y la sección 2.3. Comedi_test para la USBDUX-sigma.

4.2.3 Adquisición o emisión de señal analógica

Las tarjetas DAQ USBDUX están diseñadas para realizar adquisición de señales analógicas de forma síncrona o asíncrona.

En la adquisición síncrona el proceso es temporizado por un reloj global, que gestiona y controla la tarjeta y el resto de los componentes del sistema, es decir existe una sincronización temporal entre la tarjeta y el PC.

RTAI es la plataforma elegida para la elaboración de este proyecto por todas las razones explicadas anteriormente. RTAI ofrece un preciso servicio de temporización para sus tareas.
Gracias a este sincronismo entre a tarjeta DAQ USBDUX y el PC gobernado por RTAI y gracias a las latencias fijas de las funciones ofrecidas en Comedi de simple adquisición/emisión se pueden realizar estimulaciones/adquisiciones de alta precisión del orden de los milisegundos

En la adquisición de forma asíncrona, modo en el que la tarjeta realiza su captación de datos sin ninguna sincronización con el PC, realiza su tarea y entregan en ese momento los datos al PC. Esto conlleva a que las restricciones temporales de adquisición se los impone la tarjeta DAQ a si misma obteniéndose una gran potencia de muestreo. No es necesario que la gestión se realice bajo un control estricto temporal por lo que esta tarjeta se utilizara en un GPOS, en concreto un Ubuntu 12.02

La adquisición asíncrona de muestras de señal analógica que se realiza por medio de comandos, y la DAQ USBDUX-fast se utilizan para monitorizar y grabar la actuación del sistema completo. El testeo de la eficacia y límites del sistema se realizara a través de los datos que recoja esta tarjeta.

Comedi en su funcionalidad en RTAI, Kcomedilib, ofrece unas funciones o herramientas que posibilitan un control temporal del sistema y por tanto sincronismo sobre una configuración asíncrona del sistema. En los apartados de adquisición o emisión asíncrona sobre RTAI se explican estos métodos.

4.2.3.1 Adquisición o emisión síncrona de señal analógica.

La configuración del sistema se realiza de manera síncrona, la DAQ responde a peticiones realizadas y controladas por el PC. Las DAQ'S pueden ser utilizadas en GPOS y RTOS. La temporización y sincronización de los procesos en cada tipo de sistema operativo será realizada de manera diferente y ofrecerá posibilidades y limites muy diferentes.

La adquisición o emisión síncrona de alta precisión se realiza en RTAI por todas las razones discutidas a lo largo de todo el proyecto. RTAI ofrece sus propios relojes y sistemas de temporización que son empleados para la adquisición o emisión controlada de señales analógicas. Se utilizan los conceptos obtenidos de la documentación de RTAI disponible en www.cs.ru.nl/lab/rtai/.

Es necesaria la librería rtai_comedi.h para el manejo de las funciones de Comedi en RTAI y la librería rtai_lxrt.h para el manejo de las tareas RTAI y demás procesos en modo usuario.

La adquisición/emisión de alta precisión en esta configuración síncrona se logra por medio de la implementación de las instrucciones síncronas de adquisición/emisión de una simple muestra en una tarea periódica RTAI de máxima prioridad. La configuración del modo de velocidad del protocolo USB 2.0 condiciona la latencia mínima de la tarea periódica ya que condiciona la latencia de bloqueo de la tarea o proceso que realice la petición.El sistema síncrono usa el siguiente sistema de temporización de RTAI.

```
rt_set_oneshot_mode();
  task_period_count = nano2count(task_period_ns);
  timer_period_count = start_rt_timer(task_period_count);
```

La periodicidad de las tareas en el sistema se logra gracias al system time que determina la frecuencia de actuación del planificador y por tanto lo tanto el tiempo de uso de la CPU por un hilo o proceso en un momento concreto del tiempo. El system time es de 10^6 ordenes de unidades menor que las latencias que manejamos en la adquisición síncrona.

La periodicidad de las tareas de adquisición o emisión de valores analógicos se realiza por medio de la llamada a las siguientes funciones. La instrucción `rt_task_make_periodic` hace periódica la tarea dese un origen. Para la sincronización de varias tareas es necesario fijar un origen de tiempos común que se almacena en una variable global del sistema. La tarea se realiza en un bucle infinito en el que tras realizar su actuación de diseño llama a la función `rt_task_wait_period();`

```
origen= rt_get_time() ;
rt_task_make_periodic(task,origen+ period_count,period_count);
while(1){
      Realiza acción. Ejemplo, comedi_data_read
rt_task_wait_period();
}
```

En el caso de lo GPOS la adquisición síncrona se realiza por medio de un bucle infinito en el que se van adquiriendo datos respecto a un condicionamiento temporal impuesto por el sistema operativo. El manejo de las constantes temporal en los sistemas operativos de propósito general es impredecible y no ofrece un comportamiento estricto. La función utilizada para suspender y dotarle de cierta periodicidad al proceso es el comando usleeep la cual suspende la ejecución durante un intervalo de tiempo medido en microsegundo.

También comedi nos ofrece una función que permite la adquisición simple con un retraso impuesto, comedi_data_read_delayed.

4.2.3.2 Adquisición asíncrona

En la adquisición asíncrona la temporización la impone la tarjeta DAQ USBDUX. El control del tiempo por parte del PC puede ser menos estricto. El uso de la adquisición asíncrona de Comedi, en espacio usuario Comedilib, ofrece buenos resultados en la elaboración de control de ciclo abierto en los que solo se realiza adquisición o emisión en una dirección sobre cualquier PC actual sin necesidad de sistema operativo en tiempo real. La transferencia de datos desde la tarjeta al PC se realiza a través de USB a buenas velocidades. En aplicaciones que no sea crítico el tiempo de recepción de los datos es una

herramienta muy potente. En sistemas con RTAI y a través de Kcomedilib se posibilita la elaboración de un sistema síncrono sobre esa configuración asíncrona.

4.2.3.2.1 Adquisición asíncrona en Comedilib

La adquisición asíncrona con las funciones de Comedilib se realiza sobre el PC con sistema operativo de propósito general Ubuntu 12.02, por medio las DAQ USBDUX. En concreto se realiza por medio de la configuración de un comando Comedi sobre la tarjeta DAQ USBDUX-fast en el que en su estructura se fija los parámetros necesarios para realizar la grabación de las pruebas de que se realicen desde la DAQ USBDUX-sigma. El código empleado se encuentra en el anexo, se trata del programa de prueba 1.4. Programa de adquisición asíncrona en modo usuario con comedilib. Este programa es un ejemplo ofrecido en la página del fabricante cmd.c.

La información completa de los diferentes componentes de la tarjeta, la funcionalidad que tienen y las posibles configuraciones que manejan así como demás información relevante ofrecida por comedi_test para la ejecución asíncrona en la tarjeta DAQ USBDUX-fast se encuentra en el Anexo (véase Anexo: Manual del programador, 1.3.comedi_test USBDUX-fast).

Un ejemplo de las opciones que habría que fijar para ejecutar el programa y que por medio de comando, se realiza la adquisición asíncrona en background y por medio de secuencias periódicas de 1000 adquisiciones del canal 0, son las siguientes:

`options.filename = "/dev/comedi0";`	Fichero device
`options.subdevice = 0;`	Analog input
`options.channel = 0;`	Canal 0 (nombre)
`options.range = 0;`	[-0.75:0.75]
`options.aref = AREF_GROUND;`	Referencia a tierra
`options.n_chan = 1;`	Cantidad canals
`options.n_scan = 1000;`	# secuencias
`options.freq = 60000.0;`	Frecuencia Hz

Tabla 8: Ejemplo configuración parámetros para la ejecución de comandos en programa cmd.c

El programa genera una marca de tiempos antes de comenzar la adquisición asíncrona y otra al acabar la tarea. El programa reporta el tiempo de ejecución de la adquisición asíncrona con una precisión de microsegundos.

Como la tarjeta se emplea en un GPOS se utiliza la instrucción de comedilib:

```
comedi_get_cmd_generic_timed(dev, subdevice, cmd, n_chan,
scan_period_nanosec);
```

Con esta instrucción permitimos que sea comedilib el que cree un comando estanco preparado para soportar la tasa de muestreo referenciada por scan_period_nanosec.

Cada device conectado requiere de unos parámetros para la configuración de su comando, la DAQ USBDUX-fast difiere en la funcionalidad de la USBDUX-sigma, de nuevo se

aconseja realizar un comedi_test para poder observar las banderas de funcionalidad o tiempos mínimos requeridos. Los comandos pueden ser configurados manualmente por medio de los parámetros ofrecidos por comedi_test. La salida de comedi_test con parámetros de configuración de comandos para la tarjeta DAQ USBDUX-sigma se encuentra en el Anexo (véase Anexo: Manual del programador, 2.3.comedi_test USBDUX-sigma).

Los datos son enviados desde la tarjeta DAQ al PC por medio del protocolo USB en modo isócrono de transferencia. En sistemas de propósito general, Comedi en espacio usuario no ofrece funciones que permitan el manejo de una muestra simple analógica manipulada por comandos. Son las funciones estándar de Linux read y write quien realizan la adquisición de un buffer de hasta un numero de muestras configurable. En RTAI encontramos soluciones y herramientas para dar sincronismo al mecanismo asíncrono.

Otra alternativa es Comedirecord. Comedirecod es un programa osciloscopio que realiza también adquisición asíncrona. Posee una ventana en la que se dibuja la función adquirida. La tasa de muestro se ve reducida respecto al programa cmd.c por esta razón.

Las muestras adquiridas en ambos casos son guardadas en ficheros de texto con formato .dat en el que por medio de columna tabulada graba los datos adquiridos en cada canal de manera secuencial. La primera columna representa el número de scan, la demás columnas representan el canal analógico empleado.

4.2.3.2.2 Adquisición asíncrona en RTAI

Como se decía anterior mente la adquisición asíncrona puede ser empleada para la realización de un sistema síncrono, gracias a la precisión temporal ofrecida por la tarjeta DAQ, la velocidad del protocolo USB y a las funciones de señalización ofrecidas en RTAI y Comedi que posibilitan la adquisición/emisión de una muestra simple de señal analógica con la tarjeta configurada para realizar un comando o secuencia continua de adqusiscion en serie de canales analógicos de señal. Estas funciones están explicadas en los README del proyecto RTAI y existen ejemplos de ello en RTAI/showroom.

Las instrucciones de Kcomedilib que dotan de sincronismo al sistema asíncrono son rt_comedi_commad_data_read() y algunas variantes que añaden control temporal o control de banderas.

```
long rt_comedi_command_data_wread(void *dev, unsigned int subdev, long
nchans, lsampl_t *data, long *mask);
```

- Condicional a la epera del semaforo
  `rt_comedi_command_data_wread_if`
- Condicional a espera el semáforo hasta un tiempo determinado
  `rt_comedi_command_data_wread_until`
- Condicional a espera el semáforo de manera periódica.
  `rt_comedi_command_data_wread_timed`

Por medio de un comando que configura la tarjeta a una frecuencia de adquisición continua y precisa de frecuencia configurable, se muestrea la señal analógica a la entrada del canal configurado. La función rt_comedi_register_callback, habilita un semáforo para el sistema de adquisición.

```
int rt_comedi_register_callback(void *dev, unsigned int subdev, unsigned
int  mask, int (*callback)(unsigned int, void *), void *task);
```

Por compatibilidad con Comedi la función en modo usuario el argumento callback se fija en NULL, el argumento task especifica la tarea que se lanzara a la espera de la llegada asíncrona. Este semáforo determina el momento de llegada de muestra al PC. El sistema se inicializa en modo high-speed. El protocolo USB en modo high-speed trabaja con latencias fijas de 125 microsegundos (microframe). En una tarea RTAI de máxima prioridad se obtienen muestras cada vez que el semáforo permute, indicando la llegada del dato o muestra adquirida. La tarea se bloque a la espera del siguiente por medio de las funciones de espera habilitadas en RTAI, por ejemplo rt_comedi_wait. La DAQ USBDUX se configura por medio de un comando (cmd) a una tasa de muestreo superior a varios microframe (1 milisegundo por ejemplo), para hacer menos crítico el transvase de información de la tarjeta DAQ USBDUX al PC. El Pseudocódigo de este mecanismo es el siguiente:

```
rt_comedi_register_callback(dev,subdev,COMEDI_CB_EOS,NULL,task);
        do_cmd();
        while(1){
        rt_comedi_wait(&val);
          if (val & COMEDI_CB_EOS)
          rt_comedi_command_data_read(dev, subdev, NCHAN, data);
         }
```

El código de ejemplo de adquisición asíncrona en modo usuario a través de Comedi instalado en un sistema RTAI se encuentra en Anexo (véase Anexo: Manual del programador, 2.4.6.Adquisición RTAI asíncrono).

4.2.3.2.3 Emisión asíncrona en RTAI

En el caso de la emisión de señal es posible configurar el sistema por medio de semáforos o banderas que habilitan el envío de datos del PC a la tarjeta DAQ USBDUX con una latencia fija similar a la tasa de muestreo de emisión de señal fijada por medio de la configuración de un comando asíncrono de emisión de señal. Por medio de las señales de temporización de RTAI, que bloquean la tarea un tiempo fijo o por medio de la implementación de una tarea periódica se realiza el envío a través de rt_comedi_commad_data_write.

```
long rt_comedi_command_data_write(void *dev, unsigned int subdev, long
nchans, lsampl_t *data);
```

La señal a emitir se codifica en una cadena, la cual se envía dato a dato. La función rt_comedi_trigger especifica el inicio de la emisión de la cadena, activa la ejecución.

```
int rt_comedi_trigger(void *dev, unsigned int subdev);
```

El código de ejemplo se encuentra en el Anexo (véase Anexo: Manual del programador, 2.4.5.Emisión RTAI asíncrono).

4.3 Ciclo cerrado de estimulación dependiente de la actividad

La elaboración del ciclo cerrado se realiza sobre una tarea periódica en el que se realizan de manera secuencial la función de adquisición de muestra, fase de detección de evento y decisión de estimulación y, por último, fase de estimulación si procede.

Las tarjetas DAQ USBDUX se comunican con el PC a través de las pilas USB del kernel genérico de Linux, esto provoca que la actuación sobre la tarjeta se realice por medio de instrucciones que maneja el sistema operativo Linux, por lo que no es adecuado la utilización de tareas hard real time. Al tomar el control Linux en la actuación respecto a la tarjeta la temporización del sistema queda en manos de nuevo sobre un sistema en el que no se puede asegurar las condiciones temporales estrictas impuestas sobre el mismo.

El sistema de ciclos cerrados, al igual que la adquisición/emisión se puede implementar a través de las funciones síncronas o asíncronas de manipulación de muestras analógicas que ofrece Comedi. Ambos sistemas se implementan con las siguientes premisas:

- Los ciclos se ejecutan sobre tareas soft real time. Es necesario la utilización de las librerías rtai_lxrt para poder programar la aplicación con los objetos y funciones que se manejan en espacio usuario o para poder manejar comedilib.

- Para evitar que se produzcan fallos por los refrescos de memoria y actualizaciones en el sistema de paginación se realiza un bloqueo de memoria por medio de la función mlockall ();

- La política de planificación que se elige es SCHED_FIFO.

- La necesidad de estimulación se fija en la detección de un mínimo/máximo en la función de estado de excitación del sistema que se esté analizando. Por medio de la captación de tres puntos es posible determinar un cambio de pendiente descendente y el paso a una pendiente ascendente. La tasa de muestreo de adquisición de estas muestras está determinada por la periodicidad del ciclo. En cada periodo de la tarea RTAI se adquiere una muestra la cual es comparada con otras dos muestras obtenidas anteriormente. Si la muestra obtenida es mayor que la anterior y a su vez la muestra anterior tiene un valor inferior a la adquirida dos periodos atrás nos encontramos con un momento de necesidad de estimulación. Este es el algoritmo empleado para determinar la emisión de señal analógica. Este mecanismo de detección es el más sencillo posible para detectar mínimos en la función. El algoritmo de detección puede ser todo lo restrictivo o complicado que se desee o necesite, el problema es que la ejecución de más instrucciones en la tarea periódica conlleva una ampliación del periodo mínimo que el ciclo puede manejar.

4.3.1 Configuración síncrona del sistema para la implementación de ciclos cerrados

Todas las acciones que se realicen sobre la tarjeta DAQ USBDUX-sigma se realizan de manera síncrona y bajo el dominio temporal impuesto por los relojes de RTAI. Con una periodicidad fijada desde RTAI por medio de la implementación de una tarea periódica se elabora un sistema de ejecución cíclica.

La temporización y la periodicidad se realizan al igual que lo explicado anteriormente para la adquisición o estimulación síncrona. En el anexo se incluye el código desarrollado para la obtención de los ciclos cerrados (véase Anexo: Manual del programador, 2.4.7. Ciclo cerrado configuración síncrona).

Las condiciones temporales se imponen de manera esquemática por medio de la elaboración de la traza de tiempos de ejecución de las instrucciones de la tarea RTAI y se fija así su periodo mínimo.

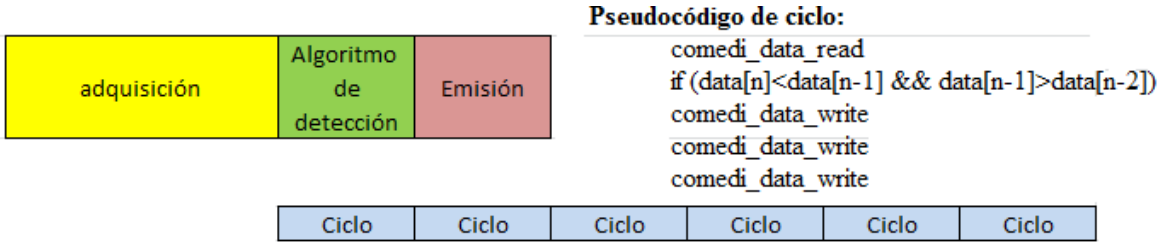


Figura 22: traza de tiempos de las tareas y pseudocódigo que componen un ciclo, y representación esquemática de la ejecución cíclica. Algoritmo de detección de mininos en la señal adquirida.

En la configuración síncrona el nivel de complejidad de la estimulación y del algoritmo de detección de eventos determina la periodicidad del ciclo. Las instrucciones de adquisición síncronas de Comedi en el driver de la DAQ USBDUX-sigma están implementadas en funciones que ejecutan un comando asíncrono de alta precisión que requieren de 2 ms de latencia fija.

4.3.2 Configuración mixta del sistema para la implementación de ciclos cerrados

En esta configuración se utilizara la potencia y flexibilidad de adquisición que ofrece la tarjeta DAQ USBDUX-sigma, la cual será la encargada determinar el periodo mínimo del ciclo. Con la adquisición configurable a través de los comandos de Comedi, modo de adquisición que utiliza el modo de transferencia USB que ofrece garantías temporales de tiempo real soft, latencia fija de transmisión (microframe) de muestras desde la DAQ al PC peo posibilidad de pérdidas de datos o fallos en transmisión (no corrector de errores), se posibilita crear un sistema de precisión menor de ciclo que la ofrecida en la configuración síncrona. Las tarjetas son configurables hasta 1 kHz de tasa de muestreo en esta configuración sobre RTAI con las funciones de Kcomedilib. En Kcomedilib encontramos el mecanismo que nos permite dar cierto sincronismo al sistema asíncrono que configuran los comandos y permite la adquisición de una muestra simple adquirida por la DAQ,

comedi_comand_data_read. Las tarjetas adquieren a una tasa fija y envían el dato al PC por medio del protocolo USB. Por medio de los sistemas de señalización que ofrece RTAI se identifica la llegada de un nuevo dato al PC. Las señales de semáforo de RTAI que regula la llegada de un dato nuevo al PC permite crear un ciclo en esa ventana o espacio de tiempo medio fijo que se da entre la adquisición de dos muestras. Estas ventanas temporales son como máximo igual al periodo que fija la tasa de adquisición. Entre cada adquisición se ejecuta el algoritmo de detección y la estimulación si procede. La estimulación puede ser una tarea que se ejecuta entre muestras adquiridas en el caso de que se detecte el objetivo o simple funciones síncronas que realicen la estimulación. La precisión del sistema que se fija en 500 microsegundos, las tasas de adquisición son múltiplos de este valor.

Función	**Instrucciones Pseudocódigo**	**Traza**
Bloqueo temporizado	rt_comedi_wait_timed(&val, TIMEOUT);	
Detección de nueva muestra adquirida	if (val & COMEDI_CB_EOS) {	
Lectura de muestra	rt_comedi_command_data_read(data);	
Algoritmo de detección (máximo)	if(data[n]<data[n-1] && data[n-1]> data [n-2])	
Emisión de estímulo	comedi_data_write(evento)	
	else	
Caso no detección	comedi_data_write(0)	
	}	

Tabla 9: Pseudocódigo de la configuración mixta de los ciclos cerrados de estimulación de alta precisión.

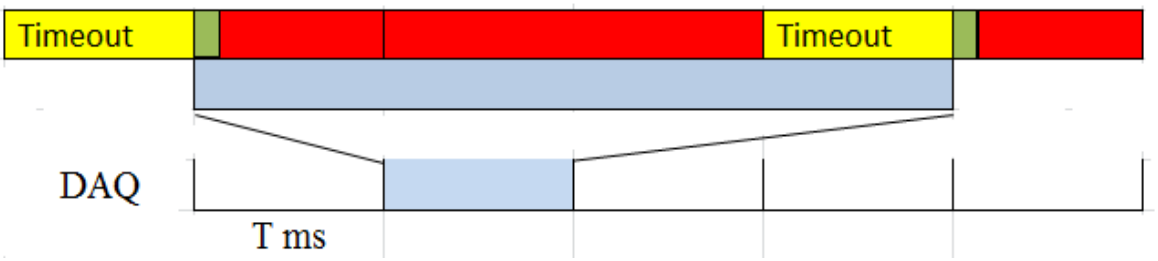


Figura 23: representación en traza de tiempos de la ejecución de los ciclos cerrados sobre el sistema en esta configuración llamada mixta.

La DAQ adquiere a una tasa con un periodo de T milisegundos. En estos T milisegundos se realiza la adquisición en un tiempo considerado prácticamente despreciable comparado con los tiempos requeridos para la emisión de estimulación. La traza en rojo representa la latencia media que se consume tanto si se produce un positivo en el algoritmo de detección de objetivo como si no. En ambos casos, las funciones que se ejecutan son funciones síncronas de Comedi para la emisión de muestras con el mismo tiempo de latencia requerido. Si no hay detección de un máximo se emite el valor que corresponda al 0 v o valor de tensión de referencia del sistema que se desea estimular. El valor de TIMEOUT es por fijado por la latencia media consumida en la estimulación.

4.3.3 Configuración asíncrona del sistema para la implementación de ciclos cerrados

Se realiza la misma implementación para la etapa de adquisición y mismo algoritmo de detección que en la configuración mixta. Solo la etapa de estimulación cambia de implementación. Se hace uso de la emisión asíncrona de señal, la cual emplea el modo de transferencia isócrono del protocolo USB el cual configura una tubería especifica con un ancho de banda fijo y por tanto con unas latencias fijas de transmisión y ofrece ciertas garantías de tiempo real que se explotan tanto en la emisión como en la adquisición. La tarjeta DAQ USBDUX-sigma se configura por medio de dos comandos de Comedi, uno que habilita el subdevice de salid analógica para emitir a la frecuencia de 1 kHz y otro para el subdevice de entrada analógica para adquirir a la misma frecuencia. El esquema del ciclo es similar al de la configuración mixta, se realiza el procesado de datos, detección de evento y estimulación, en las ventanas temporales de 1 milisegundo (frecuencia muestreo máxima 1 kHz) que ofrece la llegada de muestras adquiridas de manera asíncrona, como se observa en la figura 23.

```
if (val & COMEDI_CB_EOS) {
    data2=data1;
    data1=data;
    rt_comedi_command_data_read(dev, subdev, NCHAN, data);
          if(data<data1 && data1>data2 && data> 0X0890000){
         rt_comedi_command_data_write(dev,subdev_ao,NCHAN_AO, 1V);
         rt_sleep(nano2count(SAMP_TIME/2));

       }else{
        rt_comedi_command_data_write(dev, subdev_ao, NCHAN_AO,0V);
        rt_sleep(nano2count(SAMP_TIME/2));
    }
}
```

Pseudocódigo de la configuración asíncrona para a realización de ciclos cerrados cuyo objetivo es la emisión de señal de control de valor de tensión 1 V si se detecta un máximo en la señal que se está registrando. El código empleado entero se encuentra en el Anexo (véase Anexo: Manual del programador, 2.4.9.Ciclo cerrado configuración asíncrona).

En esta configuración es la que ofrece mayores posibilidades de temporización de mayor precisión, y por tanto ciertas garantías de tiempo real. El problema es que en esta configuración no es posible detectar la emisión de una muestra errónea. La emisión de muestras desde el PC a la DAQ o de la DAQ al PC se realiza en tiempos fijos pero con posibilidad de ocurrencia de errores. En la configuración síncrona, explicada en la sección 4.3.1 Configuración síncrona del sistema para la implementación de ciclos cerrados, o en la configuración mixta ,se emplea el modo bulk para la emisión de señal de control o estimulación, este modo de transferencia en bulk del protocolo USB no ofrece garantías de tiempo real respecto a latencias fijas de transmisión de datos pero proporciona un mecanismo por el cual es posible asegurar que la muestra que se emite es la desea, hay un control de errores. La implementación de los ciclos cerrados sobre el protocolo USB debe ser considerado soft real time para cualquier configuración empleada.

5 Integración, pruebas y resultados

En este apartado se describen las pruebas necesarias para comprobar la implementación de registro y estimulación en tiempo real con las tarjetas USBDUX. Las primeras pruebas realizadas determinan la viabilidad de ambas DAQ para realizar adquisición en configuración asíncrona. Se demuestra la potencia de registro de señal y se habilita un mecanismo preciso y configurable (tasa de muestreo y rangos de tensión de señal) para realizar registro en tiempo real por USB.

Para determinar las posibilidades y los límites en la estimulación se utiliza la DAQ USBDUX-sigma, única DAQ que permite emisión de señal. Con esta DAQ se realizan pruebas o simulaciones en varias configuraciones que nos permiten sacar varias conclusiones. Por un lado se realiza la implementación del sistema sobre las funciones y estructuras temporales que ofrece un GPOS, sistema operativo de propósito general. A su vez se realizan las mismas simulaciones sobre un sistema en tiempo real (Ubuntu con patch RTAI) con garantías temporales reales necesarias para obtener un nivel de precisión adecuado. Estas configuraciones y su comparativa nos permiten demostrar la necesidad de un sistema operativo de tiempo real para conseguir un nivel de actuación tal que permita una estimulación de precisión de milisegundos.

El protocolo USB tiene cuatro modos de transferencia de datos, en las DAQ USBDUX encontramos dos de estos modos, por ello se realizan pruebas para determinar las posibilidades y límites de cada modo de transferencia del protocolo USB que ofrecen las DAQ's. Las DAQ's USBDUX pueden interactuar con el PC por medio de funciones simples de Comedi que son implementadas en modo bulk de USB y las funciones de temporización del sistema operativo instalado en el PC, habilitando la configuración del sistema llamada síncrona. Como se describía anteriormente las DAQ pueden imponer las características temporales del sistema y realizar la emisión de señal a través de los comandos (modo isócrono USB) y gracias a las funciones de Kcomedilib de manipulación de una muestra simple se habilita la configuración asíncrona del sistema.

La imposibilidad de ejecución de las tareas RTAI que manipulan funciones del USB host en modo hard real time provoca que en la ejecución concurrente con el resto de tareas del sistema operativo Linux puedan ocurrir pérdidas de control del sistema de estimulación por necesidad de uso del procesador por parte de otra tarea concurrente de servicio. Se realizan todas las pruebas descritas anteriormente, sobre sistemas GPOS o sobre RTAI y en modo síncrono o asíncrono, en dos sistemas con recursos de procesado muy diferentes. Inicialmente se trabaja sobre un PC con un Pentium 4 a 3.2 GHz en el que se instala un kernel genérico 2.6.38 de una distribución Ubuntu 10.04 i386, versión de 32 bits, en el que se le añade un patch RTAI 3.9 que añade la funcionalidad necesaria para el manejo temporal estricto que se requiere véase la sección 5.2 Estimulación de alta precisión en un sistema implementado en un PC uniprocesador. Este PC se utiliza en los registros electrofisiológicos que realiza el GNB.

El otro PC con mayor capacidad de procesado, es un PC Intel Core i7-2600 con cuatro CPUs @ 3.40 GHz con las mismas versiones de Linux de 32 bits, RTAI, Comedi y módulo de la USBDUX-sigma. Se analizan los límites del sistema de emisión y del sistema de adquisición sobre el nuevo PC, determinando y demostrando su viabilidad, véase sección 5.3 Estimulación de alta precisión en un sistema implementado en un PC multiprocesador.

Una vez evaluadas y demostradas las características y límites que ofrecen las tarjetas USBDUX en registro en tiempo real, y dilucidadas las configuraciones para la estimulación de alta precisión (precisión de milisegundos), se realizan en este sistema PC Intel i7 las pruebas de la viabilidad de ciclos cerrados.

En un primer momento y para obtener un mayor grado de fiabilidad en las pruebas de ciclo cerrado dependiente de la actividad se realizan las pruebas sobre un generador de funciones que emite una señal sinusoidal de frecuencia regulable. La forma de evaluación de los ciclos cerrados consiste en la definición de eventos (máximos o mínimos de la función seno) para determinar el envío de señales por parte de la DAQ. Las conclusiones obtenidas se utilizan para la implementación de ciclo cerrado de estimulación dependiente de la actividad que controle la actividad de una neurona artificial electrónica. Como se ha mencionado anteriormente, esta neurona electrónica es un sistema analógico que simula la actividad de una neurona por medio de una serie de componentes hardware, emitiendo señales que expresan el estado de excitación.

Por otra parte y como se observa en la figura 24, además del PC encargado de la estimulación o los ciclos cerrados a través de la DAQ USBDUX-sigma, en todas las pruebas se utiliza un PC portátil con un Ubuntu 12.04 con Comedi y demás software necesario para el análisis del sistema al completo, incluyendo Gnuplot y octave. En este entorno, la DAQ USBDUX-fast se utiliza para grabar la emisión de señal por parte de la DAQ USBDUX que elabora la estimulación o el ciclo cerrado en el sistema objeto de estudio (generador de señales o neurona artificial) y así comprobar la precisión de los estímulos que emite la DAQ USBDUX-sigma o el correcto sincronismo del ciclo cerrado. Con la grabación de la señal se analiza los jitter que sufre el sistema, así como la latencia en la ejecución de los ciclos por medio de un programa ejecutado en Octave. En el siguiente capítulo de esta sección se analizan las características que ofrecen las tarjetas en la adquisición asíncrona de señal, que es el modo que mejor exprime las características hardware de las DAQ's, y el más adecuado en entornos con sistemas operativos de propósito general.

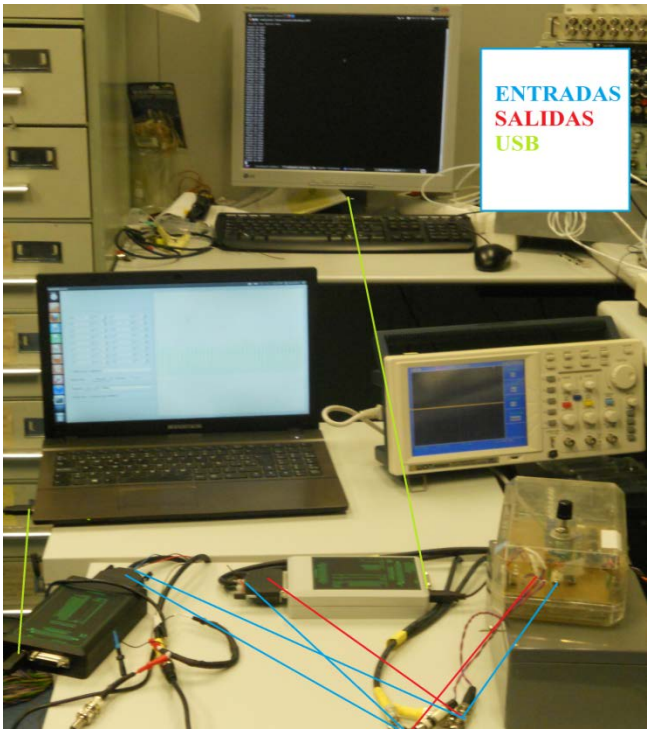


Figura 24: Integración de componentes del sistema al completo. La tabla indica el color de la dirección que toma la señal analógica, en amarillo conexiones USB 2.0. La salida de la neurona artificial se conecta a un conversor 1 a 2 BNC (splitter 'T' BNC) que deriva la señal ambas DAQ's. La DAQ USBDUX-sigma adquiere esta señal emitida por la neurona de manera cíclica. En función del objetivo, el algoritmo detector de evento determina si la estimulación. El canal de salida de la USBDUX-sigma se conecta a la entrada de la neurona artificial y otro canal de analógico de entrada de la DAQ USBDUX-fast, la cual graba ambas señales recibidas.

5.1 Registro de señalización analógica en modo asíncrono

En esta sección se describen los registros de señalización analógica emitida a través del generador de señal GF1000 a través las DAQ's USBDUX. Las adquisiciones son realizadas por medio de una configuración asíncrona del sistema PC/DAQ. Se comprueba la eficacia de la adquisición asíncrona para la implementación de sistemas en los que el volcado de datos desde la DAQ al PC no es crítico, la tasa de procesado de los datos en el PC es varias veces menor que la tasa de transferencia del protocolo USB. Este tipo de sistemas, como el osciloscopio Comedirecord o sistemas de reproducción de video o audio desde memorias flash USB, son ejemplos de sistemas de tiempo real soft. Las características de las DAQ y el modo de transferencia del protocolo USB en configuración isócrona posibilitan la creación de un sistema de registro en tiempo real soft de señal analógica. Los errores en los datos transferidos al PC pueden ocurrir, ya que el sistema carece de protocolo de control de errores en esta configuración de USB, y a su vez el sistema operativo Linux y su extensión RTAI, en el acceso a la pila USB, no ofrecen garantías de las latencias fijas y no hay posibilidad de ejecución en modo hard real time.

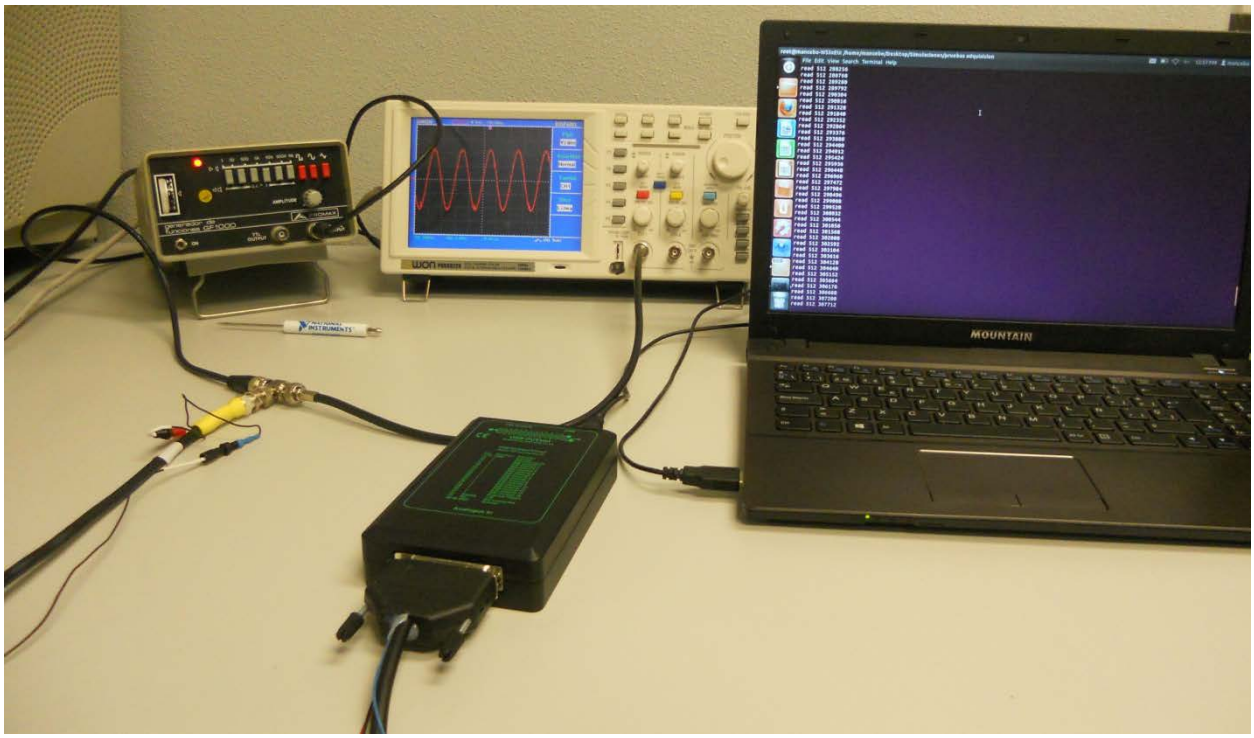


Figura 25: Montaje realizado para realizar las pruebas de adquisición. La USBDUX-fast lleva a cabo grabaciones de las señales emitidas por el generador de funciones.

5.1.1 Registro de señal analógica en modo asíncrono con Comedilib

Debido a que las condiciones temporales para la adquisición son impuestas por la DAQ y a que el volcado de datos se realiza en modo background a través del protocolo USB, tanto en GPOS como en RTAI en espacio usuario, con las funciones de Comedilib, se obtienen buenos resultados en la adquisición de señal analógica en aplicaciones open-loop. Las tarjetas DAQ USBDUX adquieren a una tasa fija configurable (cmd) con una precisión de 16 bits en la USBDUX-fast y de 24 bits en la USBDUX-sigma. Las características de las tarjetas DAQ USBDUX han sido explicadas en el diseño con detalle. Para obtener una recreación de la señal que se monitoriza la adquisición debe realizarse, si las características de la DAQ lo permiten, a 10 o más muestras por periodo de señal.

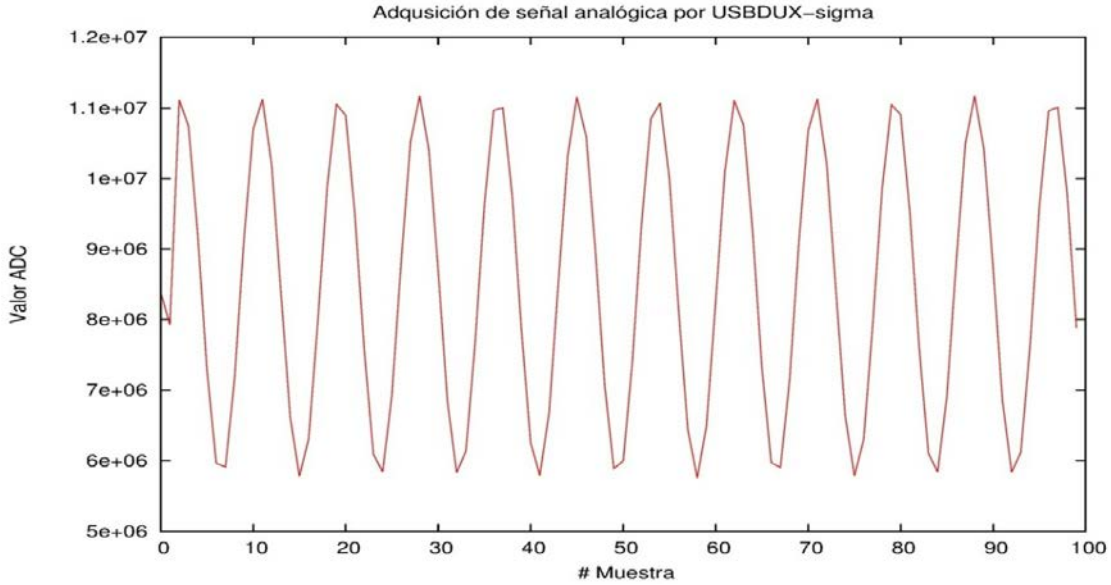


Figura 26: Ejemplo de grabación de duración 0.0125 s (100 muestras a 8 KHz) de señal producida en el generador de señales a frecuencia 200 Hz mediante la DAQ USBDUX-sigma a 8 KHz. Se observa la potencia de los conversores A/D en los valores de ADC.

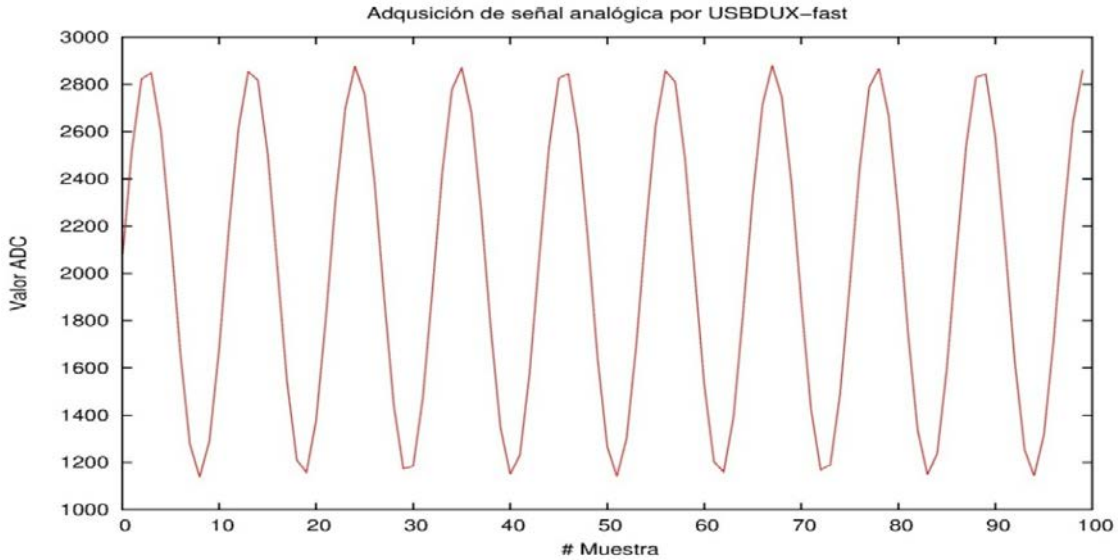


Figura 27: Grabación de una señal de 10 KHz. El comando asíncrono se configura a una frecuencia de 100Khz. En comparación con la figura 26 se observa que la precisión de los conversores A/D se reduce pero mejora la tasa de muestreo que maneja.

La precisión en la cuantificación de la señal analógico es de $1/2^{24} = 0.0596\ \mu V$. El rango de señal en voltios de los canales A/D de la USBDUX-sigma es de -1.325 a 1.325, el valor de muestra de conversor A/D, 8388608 corresponde al valor 0 V. La USBDUX-sigma ofrece menores tasas de adquisición de datos que la USBDUX-fast en igualdad de condiciones. La DAQ USBDUX-fast ofrece tasas de muestreo configurables de hasta 3 MHz para un solo canal. En modo usuario por medio de las funciones ofrecidas en Comedilib se puede hacer una conversión de los valores de muestra del conversor analógico/digital a valor de tensión. A continuación, se muestran ejemplos de grabación de 100 ms de señal emitidas por el generador de señales. Para determinar el tiempo de grabación se fija el número de muestras que se adquieren, a tasa 125 Khz se requieren 12500 muestras para obtener 100 ms.

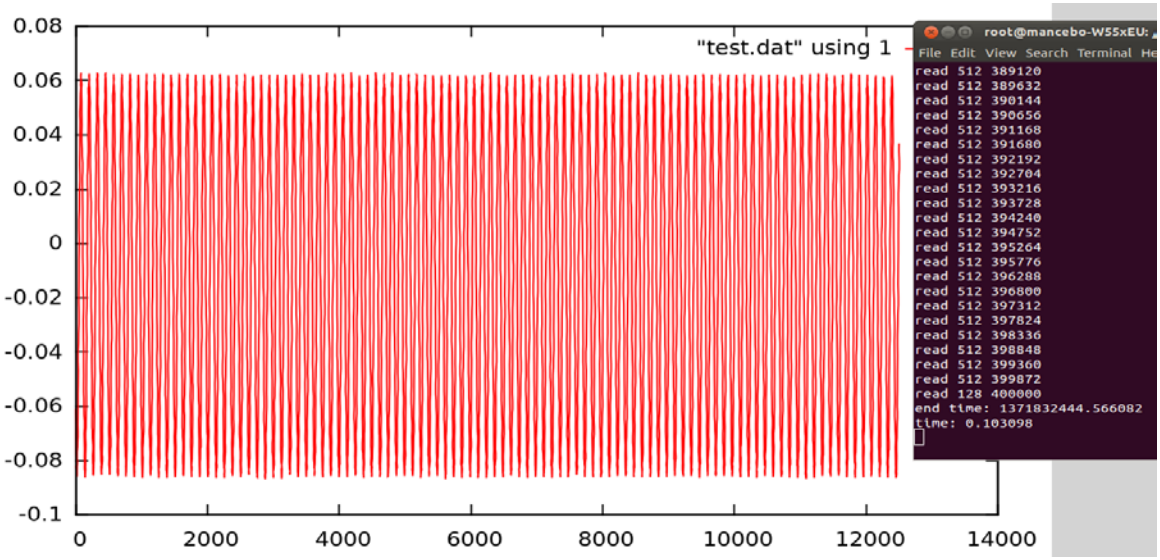


Figura 28: Adquisición de datos en el sistema Linux con tarjetas USBDUX. La aplicación Gnuplot representa los datos obtenidos de manera asíncrona por la DAQ USBDUX-fast por medio de la ejecución de un programa que configura la DAQ para realizar una adquisición a tasa fija de 125 kHz de una señal analógica de frecuencia 1kHz, realiza el volcado de datos en background y guardada en buffers de 512 bytes los datos en un fichero llamado test.dat.

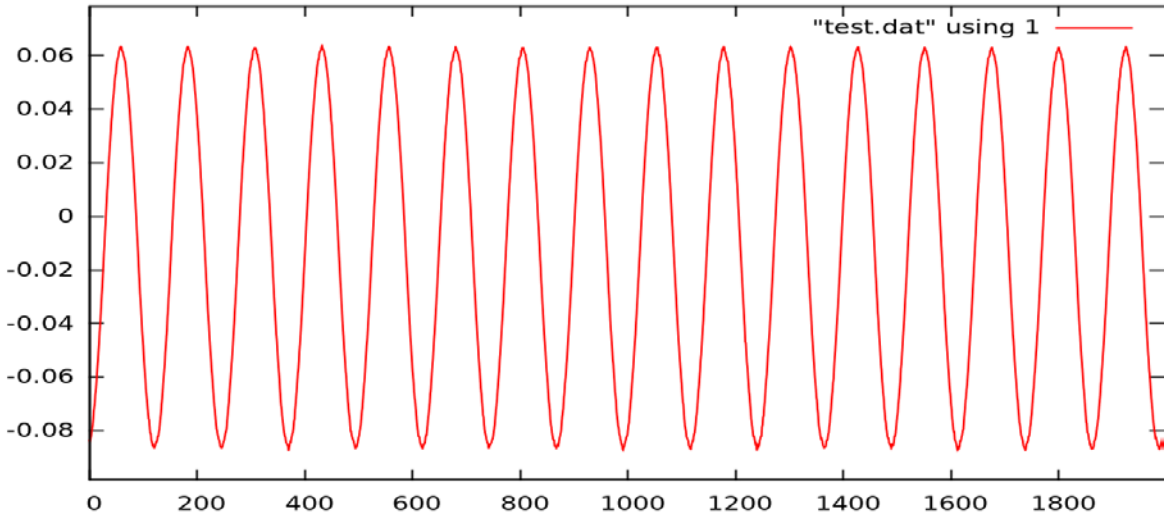


Figura 29: Gráfica realizada con los 2000 primeras muestras de la figura 28.

Más ejemplos de las grabaciones que realiza la DAQ USBDUX-fast son las figuras 32 y en adelante, se trata de figuras que muestras tramos de grabaciones de pruebas de estimulación y de ciclos cerrados. Estas pruebas son realizadas por medio de la USBDAX-fast en configuración USB 2.0 high-speed sobre un sistema operativo GPOS, Ubuntu 12.04, a una frecuencia configurable por medio de un comando de Comedi, el cual configura la tarjeta DAQ USBDUX-fast para realizar secuencias periódicas de adquisición de los canales requeridos y volcado de datos por medio de un modo de transferencia isócrono del protocolo USB, es decir en una configuración asíncrona.

5.1.2 Registro de señalización analógica en modo asíncrono en tiempo real con Kcomedilib

En esta sección se describen grabaciones de señales analógicas sobre sistemas de tiempo real en una configuración asíncrona controlada y con latencia fija gracias a las herramientas de RTAI y Kcomedilib que posibilitan la creación de un sistema de registro de señalización analógica muestra a muestra, obteniéndose así mayor nivel de sincronismo, en tiempo real soft por USB. La tarjetas DAQ's USBDUX funcionan hasta 1 kHz en tiempo real soft.

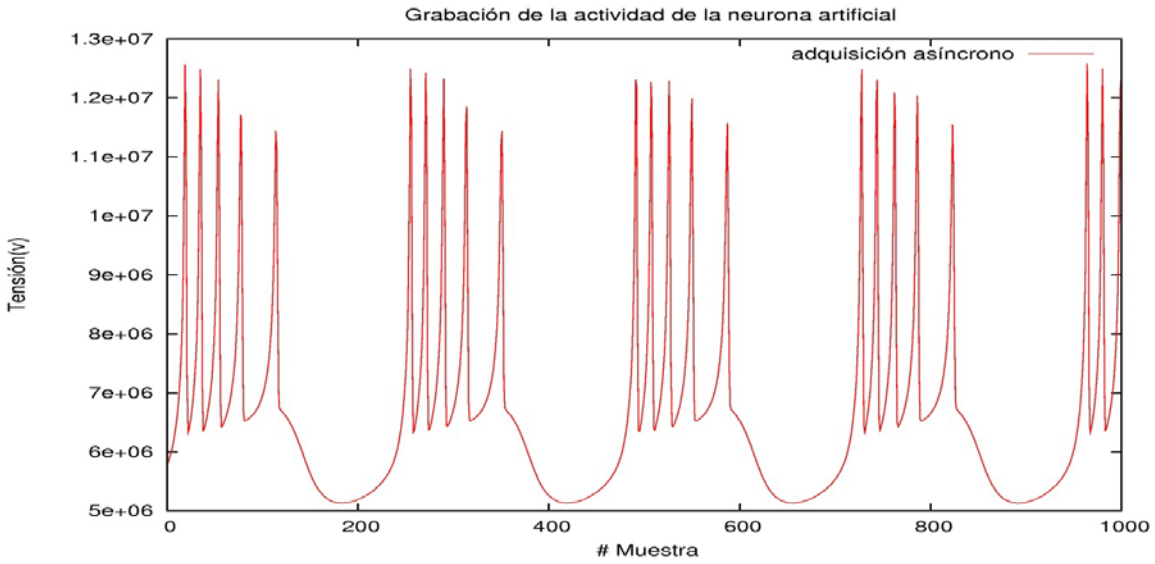


Figura 30: Adquisición de señalización analógica realizada a través de las funciones de Kcomedilib. La señal emitida desde la neurona artificial es registrada a una tasa de adquisición de 1 kHz, valor máximo de tasa de adquisición que ofrece el protocolo USB y la tarjeta DAQ USBDUX-sigma en modo asíncrono en tiempo real.

En esta configuración asíncrona, se necesita hacer una etapa de corrección del dato obtenido, y la adquisición se realiza por medio de las funciones de Kcomedilib comedi_command_data_read de dos muestras (en Comedi formato lsampl_t). La muestra completa (lsampl_t d) se obtiene de la combinación de ambas muestras:

```
rt_comedi_command_data_read(dev, subdev, NCHAN, data);
rt_comedi_command_data_read(dev, subdev, NCHAN, data +1);
if(data[0]!=data[1]){
   if(data[0]<0x000150 && dant< 0x000150)
       d=(data[0] << 16) + data[1];
   else
       d=(data[1] << 16) + data[0];
    dant=data[0];}
```

5.2 Estimulación de alta precisión en un sistema implementado en un PC uniprocesador.

Las pruebas para la emisión de señal analógica se realizan a través de la tarjeta DAQ USBDUX-sigma controlada por un PC Pentium 4 a 3.2 GHz con sistema operativo tipo Linux y Comedi instalado, y con un patch RTAI. La señal es grabada por la tarjeta USBDUX-fast configurada en modo asíncrono y el volcado de información se realiza sobre un PC portátil. La duración de la grabación se fija en 1 minuto.

5.2.1 Emisión continua de microeventos

Se desea comprobar la eficacia del sistema en la emisión de señales analógicas de alta precisión y analizar los tiempos de ejecución y jitter que se dan en la emisión de un microevento. Un microevento consiste en la emisión de una señal mínima periódica analógica, y consiste en un pulso cuadrado de potencial alto (por ejemplo 0.65 V), de un milisegundo de duración, y con un periodo de dos milisegundos, es decir una señal cuadrada de periodo dos y con un 50% de *duty cycle*.

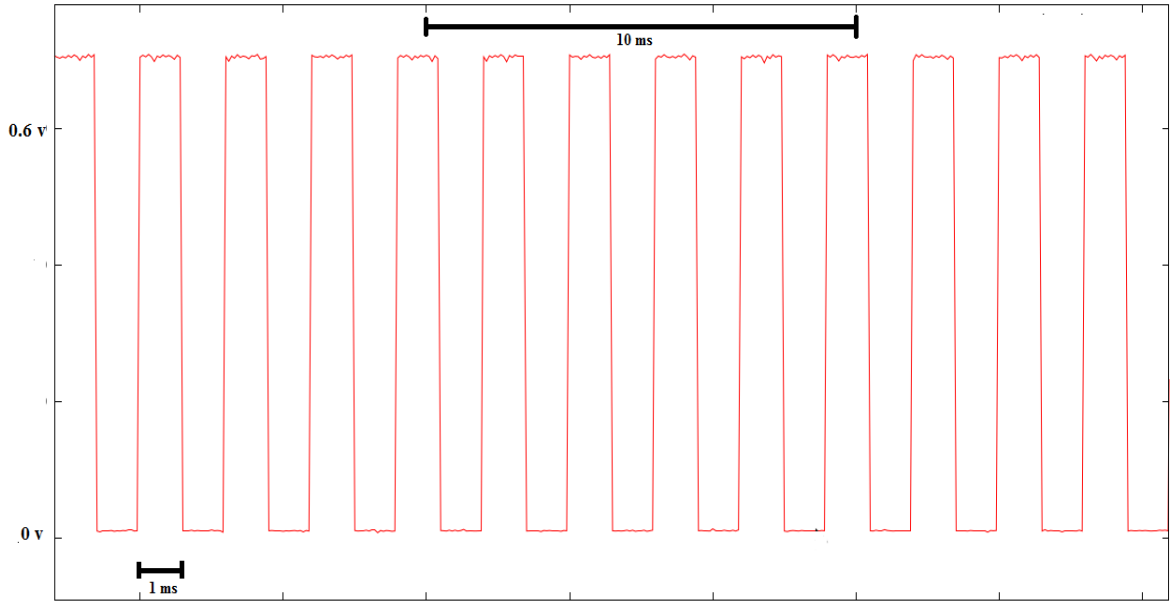


Figura 31: Grabación de estimulación: emisión continúa de microeventos. La duración del pulso que compone el microevento es de 1 milisegundo. En 10 milisegundos se ejecutan 5 microeventos.

Por medio de la emisión continua de microeventos se comprobara la eficacia del sistema bajo varias configuraciones: bajo el dominio de sistema operativo de tiempo real o de propósito general y en una configuración síncrona, donde el PC impone sus restricciones temporales o en un modo asíncrono, modo en el que la tarjeta gestiona los tiempos del sistema.

En la emisión continua de microeventos se comprueban las limitaciones en la precisión respecto a la duración del pulso y las limitaciones en los momentos de actuación requeridos para la estimulación de alta precisión temporal. Para determinar la precisión en

la duración, se analiza la duración temporal y el jitter del pulso que compone el microevento y para conocer las limitaciones en el momento de actuación se analiza la periodicidad y el jitter de los microeventos.

La emisión se implementa sobre varias tareas sobre cada tipo de sistema operativo: una tarea sobre el sistema operativo de propósito general Ubuntu 10.04, que realiza emisión síncrona de señal, y otras tres tareas sobre RTAI, dos en una configuración síncrona a velocidades de transferencia diferente, full-speed y high-speed y otra tarea que configura el sistema de forma asíncrona y a través de Kcomedilib se obtiene sincronismo gracias a la posibilidad de envío de una muestra simple por parte del PC a la tarjeta DAQ (obteniéndose un mecanismo temporal condicionado por el reloj interno de la tarjeta que consiste en un bucle while infinito que emite microeventos a frecuencia 1 kHz). Los códigos de emisión se encuentran en el Anexo (véase Anexo: Manual del programador. 2.4 Programa usados para la interactuación con la USBDUX-sigma).

Sistema operativo de propósito general

Como observamos en la figura 32 la generación de microeventos en el caso de configuración síncrona del sistema sobre un sistema operativo de propósito general requiere de un control temporal más preciso por parte del PC para realizar estimulación de alta precisión. La duración del impulso no es fija y ocurren fenómenos de pérdida del control del sistema que provocan que sea imposible realizar una estimulación precisa.

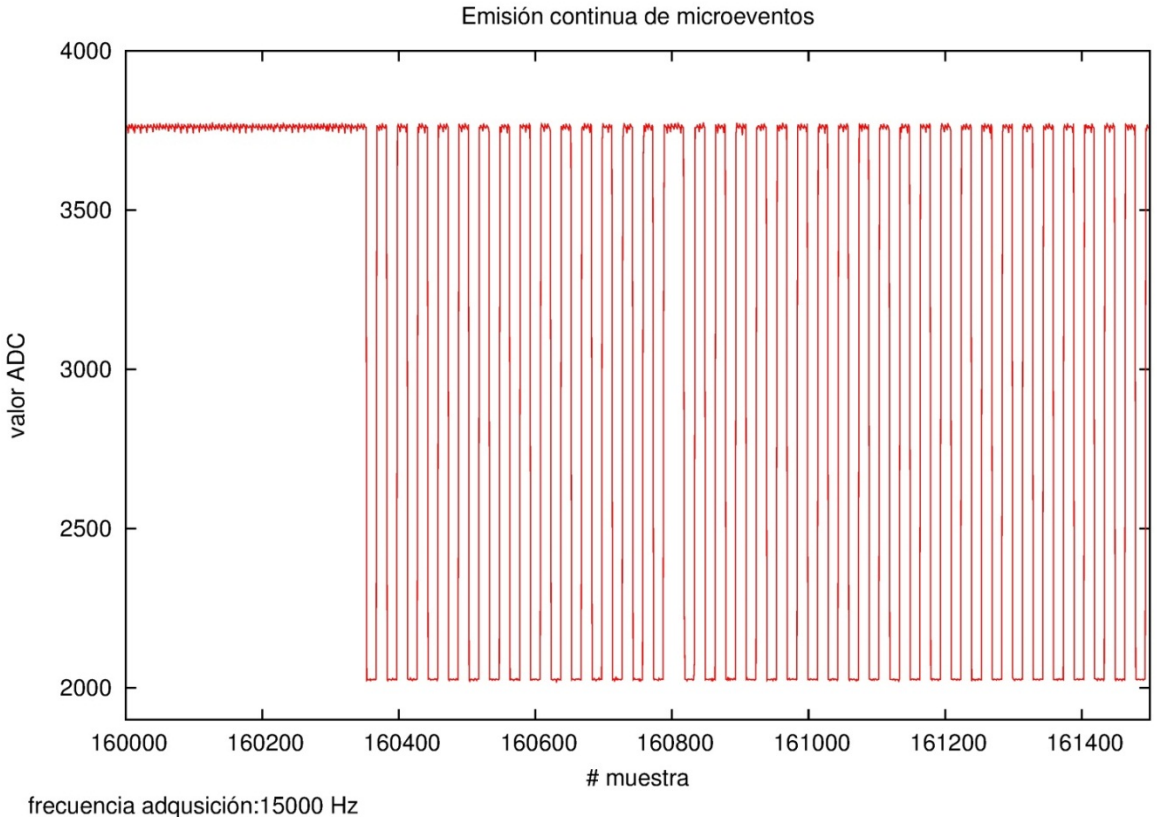


Figura 32: Grabación de señal por medio USBDUX-fast de secuencia periódica de microevento emitida por la USBDUX-sigma en un entorno de sistema operativo de propósito general. Se observa que los microeventos no tienen duración fija.

Sistema operativo en tiempo real RTAI en modo síncrono

En la configuración síncrona, el PC gracias a RTAI impone las condiciones temporales y la emisión de señal analógica se realiza por medio de la ejecución de instrucciones síncronas de emisión de una muestra simple de Comedi, bloqueando el proceso que las ejecute a una latencia fija. Las tareas RTAI se ejecutan en modo soft real time.

El microevento en una configuración síncrona y usando las instrucciones de Comedi para emisión simple depende de la configuración de la velocidad de transferencia del protocolo USB versión 2.0. Si la tarjeta está configurada en Full-speed, la velocidad está fijada en relación a un frame USB de latencia fija de 1 milisegundo. Si los componentes del PC permiten la configuración de la DAQ en high-speed, la velocidad aumenta reduciéndose la latencia a 125 microsegundos. Las instrucciones síncronas se adaptan a la velocidad de transferencia de la comunicación del PC y la DAQ. Para determinar la duración del impulso que compone el microevento se utilizan las funciones de temporización de RTAI, que mantienen en estado de suspensión el proceso que sobre la DAQ mantiene la señal en un valor de tensión el tiempo que se le indica como argumento. Para la elaboración de un microevento de frecuencia 500 Hz y de duty cicle 50%, es decir una señal de tipo pulso cuadrado de 1 milisegundo en high y 1 milisegundo en down periódico, en el caso de modo síncrono a velocidad full-speed, se utiliza la latencia fija de los frame USB y se emiten dos señales analógicas una de valor de tensión en torno a 0.65 V y la otra de 0 V. En el caso de high-speed, se utiliza un temporizador entre ambas emisiones, se fija 1 – microframe milisegundos entre cada muestra.

En una configuración full-speed en un primer análisis visual, la emisión de microeventos parece correcta, las funciones de Comedi ofrecen latencias fijas de duración:

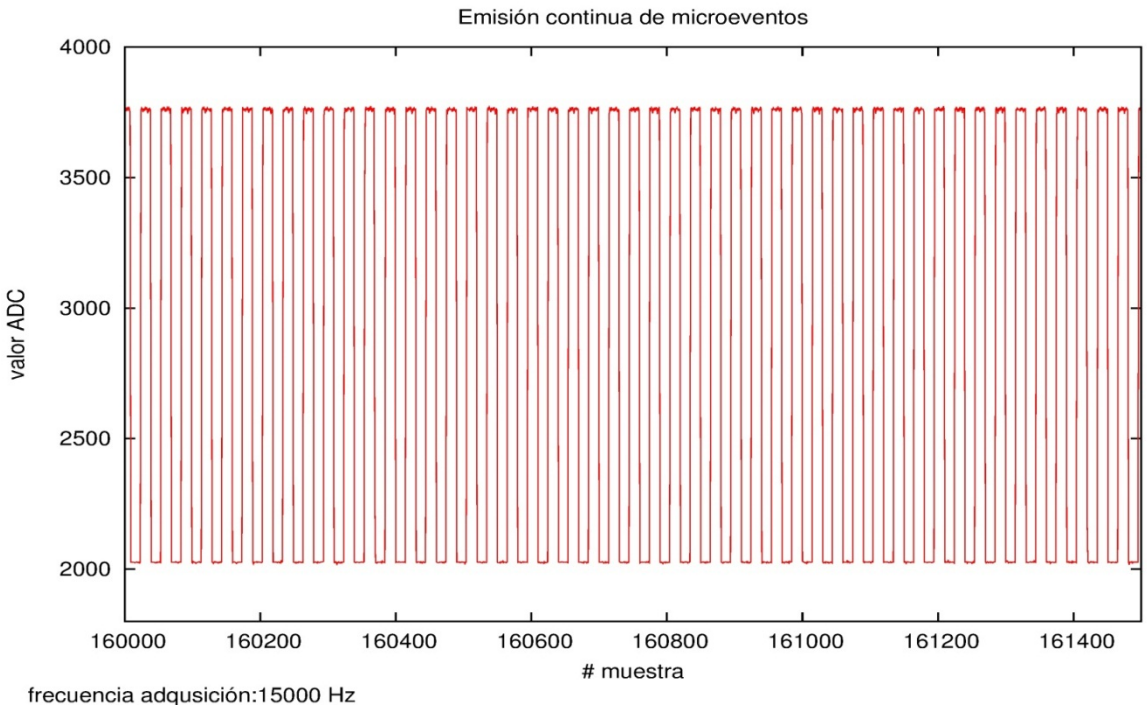


Figura 33: Grabación de señal por medio USBDUX-fast con Comedirecord de microevento emitida por la USBDUX-sigma en un entorno de tiempo real soft modo síncrono configuración USB 2.0 en full-speed. Parece que se logra una estimulación de precisión de milisegundos.

En el caso de configuración high-speed se observa un comportamiento distinto al deseado:

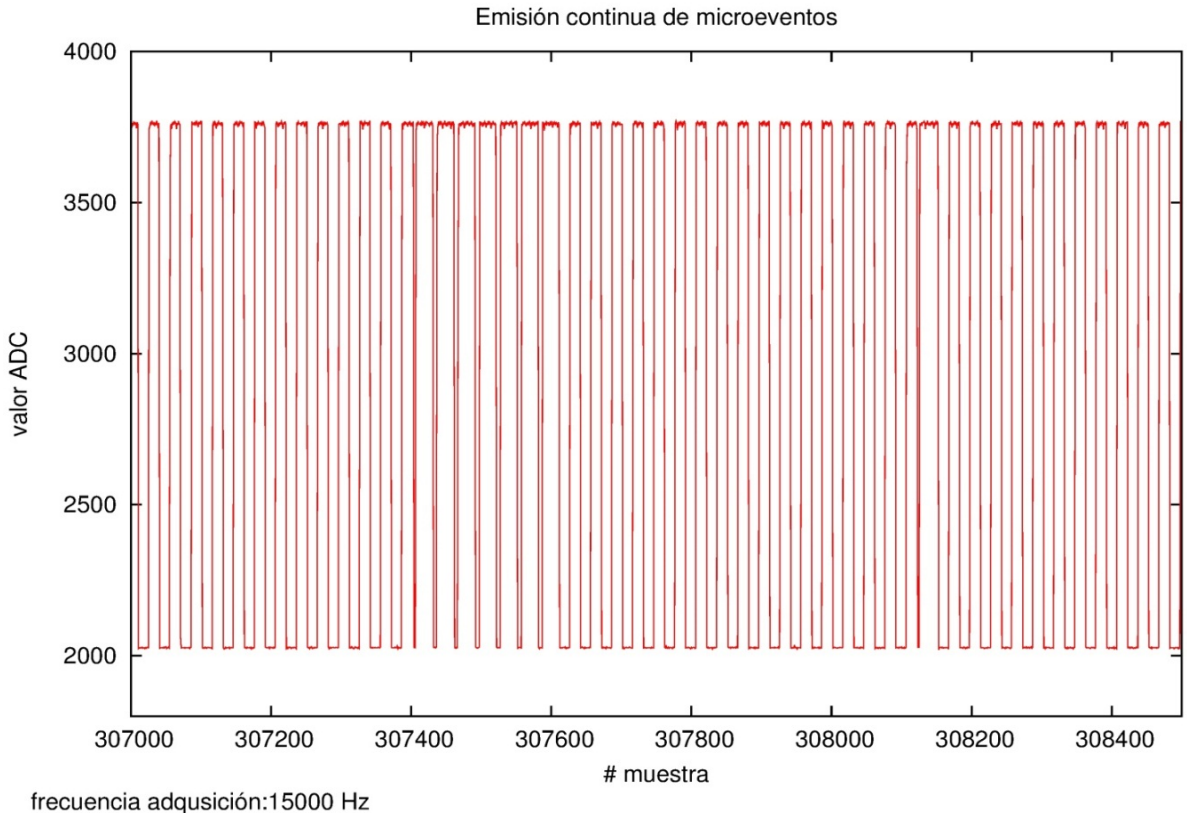


Figura 34: Grabación de señal por medio USBDUX-fast con Comedirecord de microevento emitida por la USBDUX-sigma en un entorno de tiempo real soft modo síncrono configuración USB 2.0 en high-speed. Se observa que no es posible cumplir con las características de duración de impulso impuesto.

Sobre RTAI y bajo las garantías del soft real time, respecto a precisión en la duración, hay que resaltar que el protocolo USB 2.0 en versión Full-speed ofrece un sistema más fiable sobre el control de la duración del estímulo que la configuración de velocidad high-speed. En modo high-speed la precisión la da el sistema de temporización ofrecido en RTAI, mientras que en el modo full-speed es el control de las interrupciones USB el que sincroniza e impone la duración del pulso. En el modo síncrono de USB 2.0 con configuración full-speed la duración de los impulsos parece correcta (ver Fig. 33).

Observada la ineficacia de la configuración síncrona en high-speed figura 34 en donde la duración del evento se realiza por medio de las funciones de temporización de RTAI y la implementación del sistema se realiza en un PC uniprocesador con una capacidad de recursos limitada y compartida, se descarta el uso de este modelo para la implementación de un sistema de emisión de precisión suficiente como para realizar el control de sistema, que requiera de una precisión temporal estricta, como por ejemplo los ciclos cerrados de estimulación dependiente de la actividad o gobernados por objetivo. Se decide cambiar la forma de ajustar la duración del pulso en la configuración síncrona en high-speed y se valora la necesidad de utilización de un sistema con mayor capacidad de procesado.

Sistema operativo en tiempo real RTAI en modo asíncrono

En el caso de utilizar las tarjetas DAQ USBDUX en una configuración asíncrona, los microeventos están determinados por la configuración de los comandos de Comedi, los cuales configuran la tarjeta para la emisión de señal analógica a una frecuencia determinada. La tarjeta DAQ emite el valor de tensión que es enviado por el PC. La precisión en cuanto a la duración del pulso y el momento de actuación la fija la DAQ USBDUX-sigma, solo es necesario transferir datos desde el PC a la DAQ con un ritmo que se adecue a la tasa de emisión de señal fijada. La señal estimuladora se crea offline y se guarda en un buffer. Por medio de las instrucciones de Kcomedilib se sirven los datos a la tarjeta DAQ de manera síncrona. La velocidad de transferencia juega un papel importante, el manejo de microframe en vez de frames, ofrece mejores velocidades de transferencia y coloca al sistema en una situación menos crítica en cuanto al envío de datos a través de USB. Se utiliza la tarjeta DAQ USBDUX en modo high-speed que en modo de transferencia isócrono ofrece latencias fijas de 125 microsegundos.

Para configurar un microevento similar al caso de configuración asíncrona, se configura la tarjeta DAQ USBDUX para emitir señal analógica a frecuencia 1000 Hz. La señal a emitir se configura offline sobre un buffer o array en la que cada posición de memoria representa un valor a emitir. La tarjeta se configura al inicio del programa y se queda a la espera de que le sirvan datos, la duración del array representa la duración del estímulo. Gracias a la latencia fija de la microframes USB somos capaces de servir la tarjeta DAQ de manera eficiente. El sistema de tiempo real implementado en configuración asíncrona parece realizar una estimulación precisa en duración.

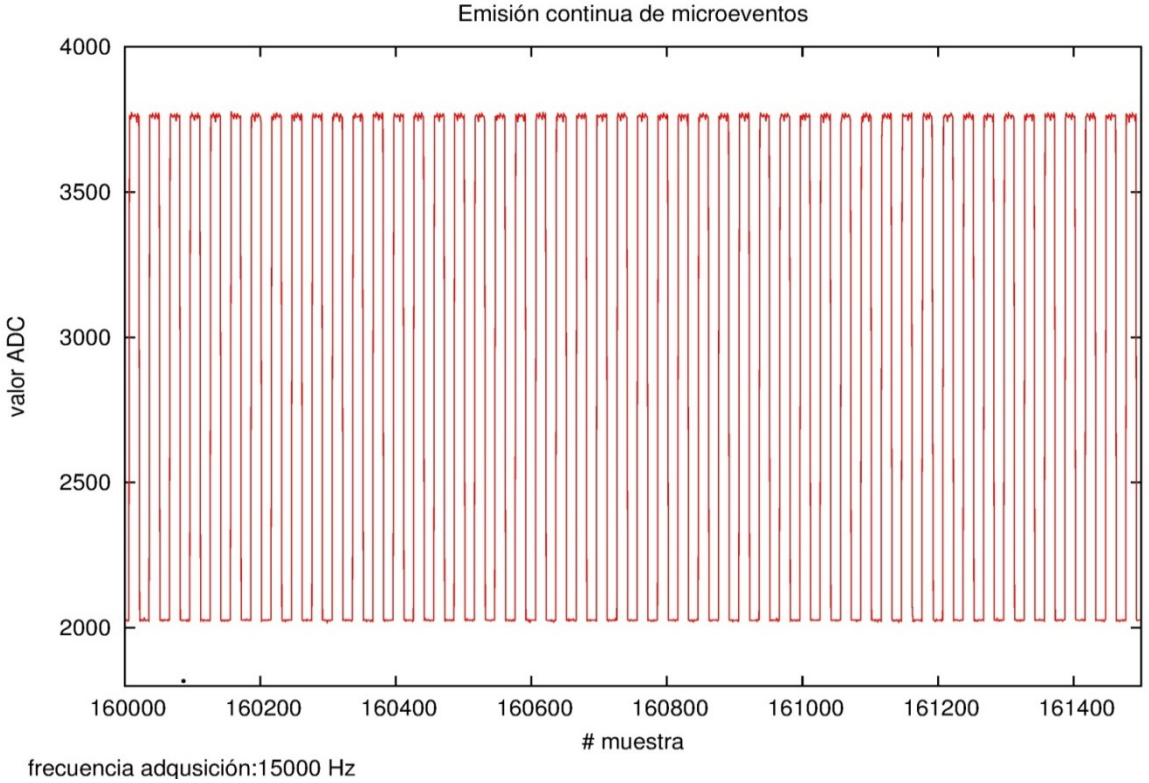


Figura 35: Grabación de señal por medio de USBDUX-fast de un microevento emitido por la USBDUX-sigma en un entorno de tiempo real soft en modo asíncrono configuración USB 2.0 en high-speed. Se observa la eficacia del envío continuo de señal analógica por parte del mecanismo isócrono que se habilita en esta configuración isócrona.

5.2.1.1 Comparativa de precisión en duración: Latencia media y jitter

En este apartado se realiza un estudio del jitter de la grabación de emisión de señal continuo y periódico para determinar la precisión en la duración del estímulo que se envía. Para ello se emplean los ficheros .dat obtenidos d la grabación de la señal analógica por medio de la USBDUX-fast a una frecuencia de 15 kHz y con 12 bits de resolución. Se contabilizan los puntos con valor mayor al umbral 0.5 V que especifica el pulso de tensión alto que compone el microevento.

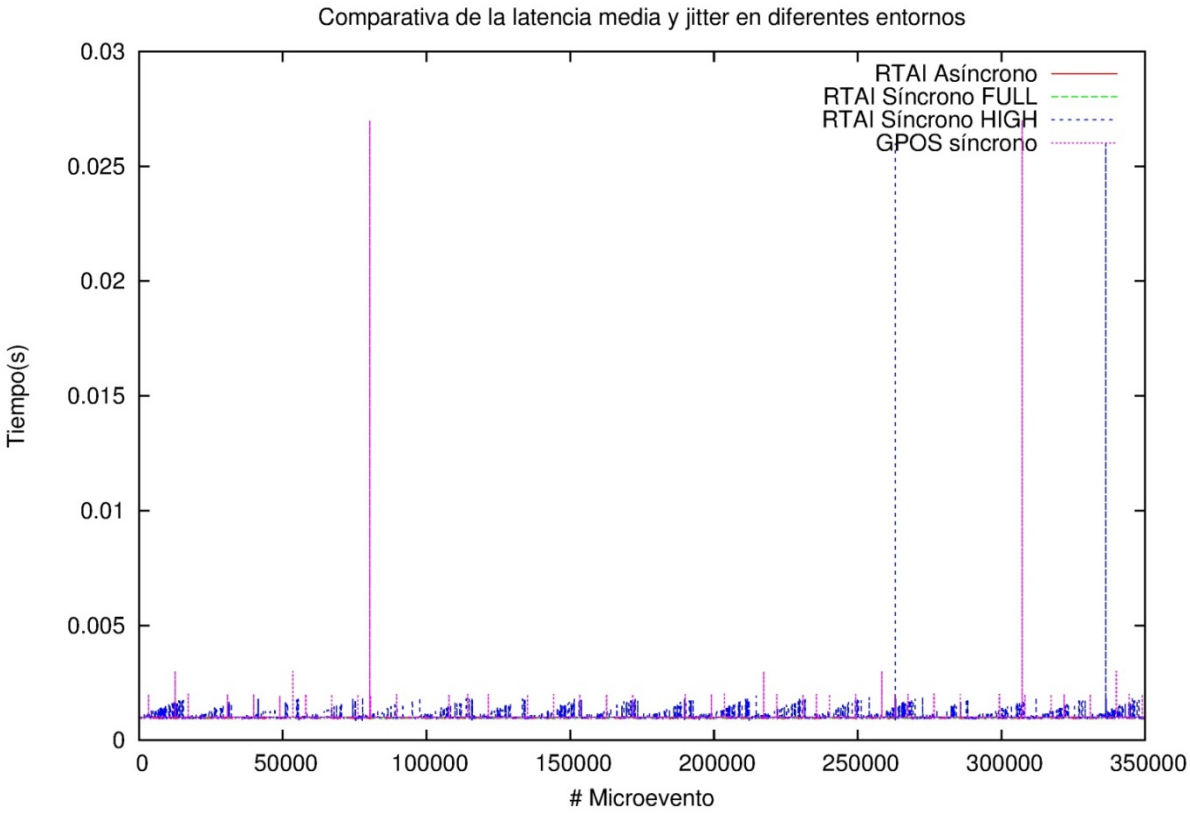


Figura 36: Gráfica comparativa de los jitter registrados en la duración del valor de tensión high que determina un microevento. El objetivo son latencias de 1 milisegundo.

Como se podia entrever, la implementacion del sistema sobre un sistema operativo de proposito general no es una alternativa para la creacion de sistemas de precision alta. Lo mismo ocurre con la configuracion USB high-speed en modo sincrono en RTAI. En la grafica comparativa figura 34 se observa el efecto registrado en la figura 36que muestra como los pulsos se alargan hasta casi el periodo completo (hasta casi un 90% de duty cycle). Esto se debe a que la temporizaion se realiza por medio de las funciones de espera que ofrece RTAI, las cuales están implementadas en modo soft y en este PC Pentium 4 de un solo núcleo no ofrecen latencias precisas.

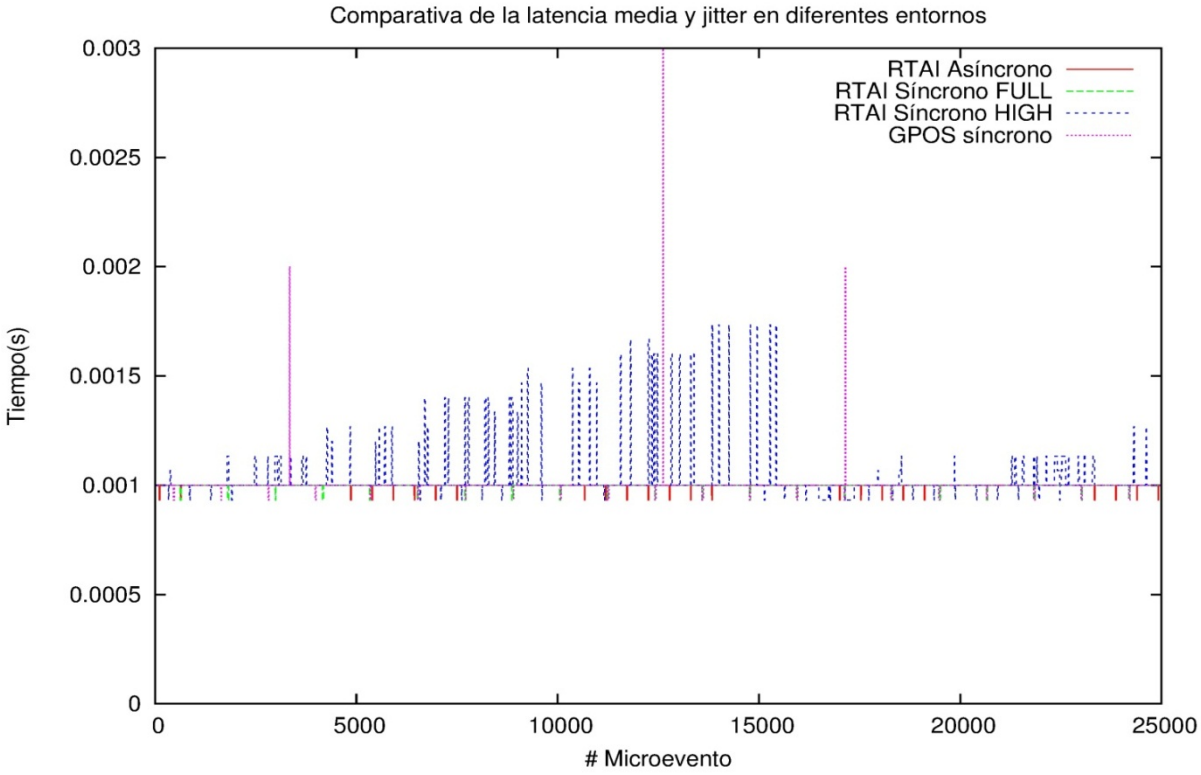


Figura 37: Ampliación de la figura 37. Comparativa de la duración del pulso de tensión alta que compone el microevento.

La duración de los pulsos emitidos por la USBDUX-sigma en un entorno RTAI en modo asíncrono y en configuración full-speed modo síncrono ofrecen buenos resultados en la emisión de microeventos con duración o latencia fija (ver Fig. 33 y Fig. 35). A continuación se observan sus jitter:

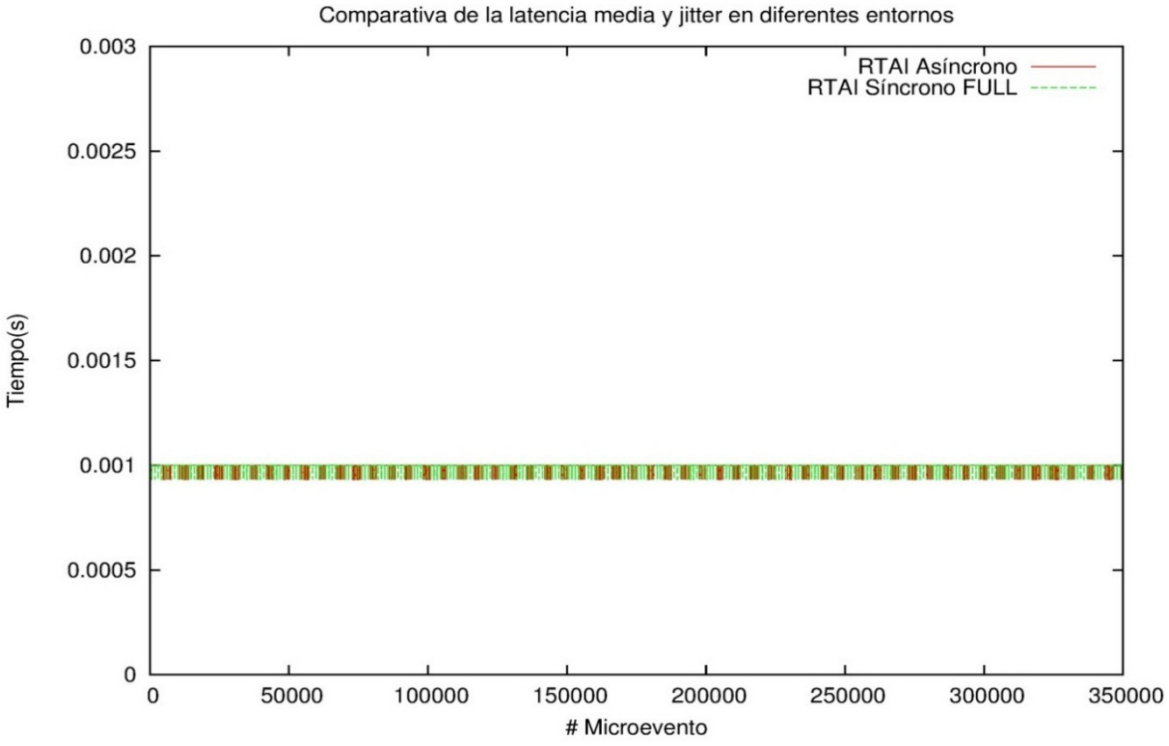


Figura 38: Comparativa de la duración del pulso de tensión alta que compone el microevento en entorno RTAI en configuración USB full-speed modo síncrono y en configuración USB high-speed modo asíncrono. En ambos casos registramos estímulos de duración high de 1 milisegundo. Parece que se puede elaborar un sistema de alta precisión.

5.2.1.2 Comparativa de precisión en momento de actuación: Latencia media y jitter

A la vista de los resultados, la viabilidad de la realización de ciclos cerrados de estimulación de alta precisión sobre una configuración de USB 2.0 a velocidad high-speed por medio de una adquisición síncrona de señal analógica es muy mala al igual que la ofrecida por los sistemas de propósito general. Si observamos la Figura 38 de nuevo pero sin los dos métodos desechados se observa la precisión en la duración del pulso que compone el microevento en las pruebas realizadas en un PC de un solo procesador.

La configuración asíncrona sobre protocolo USB 2.0 en high-speed y la síncrona sobre protocolo USB 2.0 en full-speed ambas sobre RTAI muestran resultados positivos que arrojan la posibilidad de creación de un sistema de estimulación de alta precisión temporal en duración. Para determinar la precisión temporal en el momento de actuación se analiza la periodicidad de estos impulsos.

Para determinar la periodicidad se realiza un análisis similar al utilizado para determinar la latencia en la duración del pulso que compone el microevento. Se contabiliza los puntos que componen un periodo. El código se implementa en Octave.

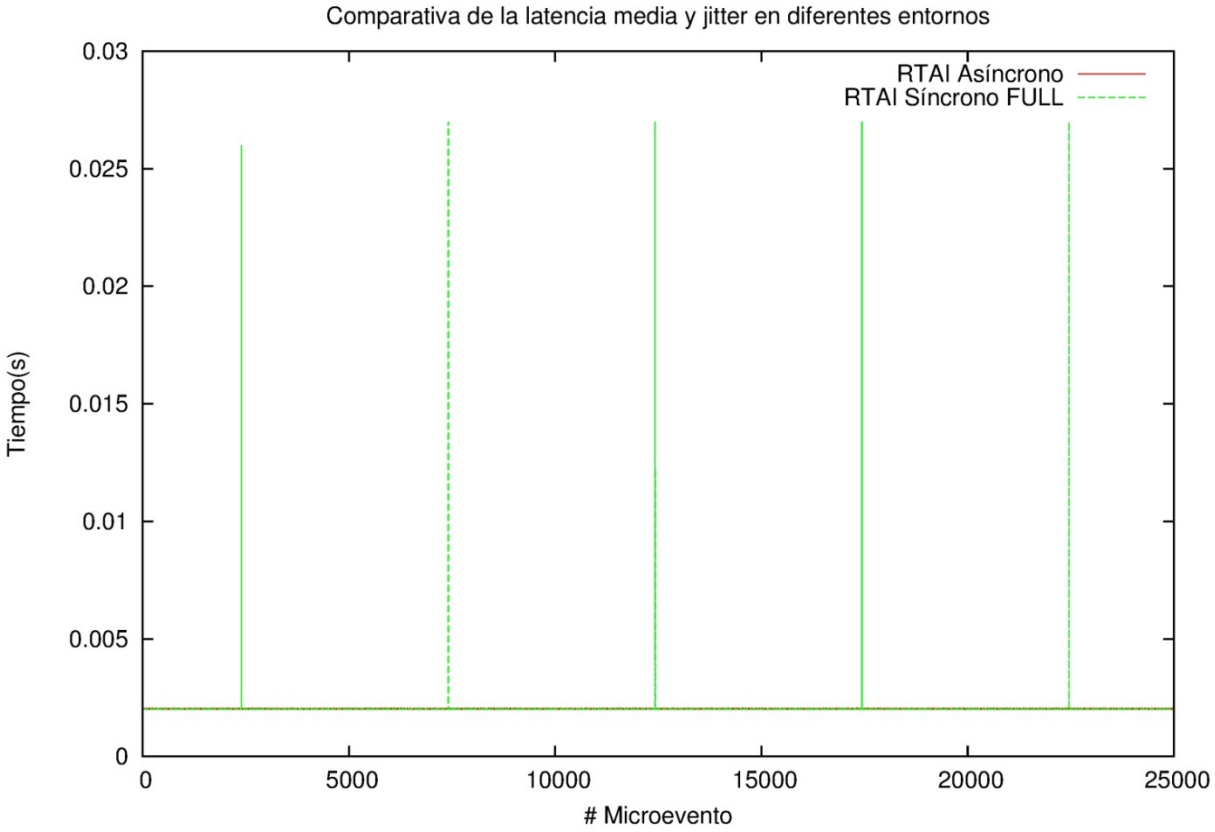


Figura 39: Jitter en la latencia de la periodicidad de emisión de un microevento, con esta grafica se analiza la precisión de actuación del ciclo estimulador, la prueba de emisión se realiza en entornos de tiempo real por medio de funciones síncronas y asíncronas para el

manejo de la tarjeta DAQ y en entornos de sistemas operativos de propósito general. El protocolo USB en modo 2.0 full-speed en las configuraciones por medio de funciones síncronas y en high-speed para configuración asíncronas.

Los resultados reflejan la inviabilidad de la emisión síncrona de señal incluso en configuración USB 2.0 Full-speed debido a que se producen perdidas de control muy elevadas, entorno a 25 milisegundos, como se observa en la figura 39.

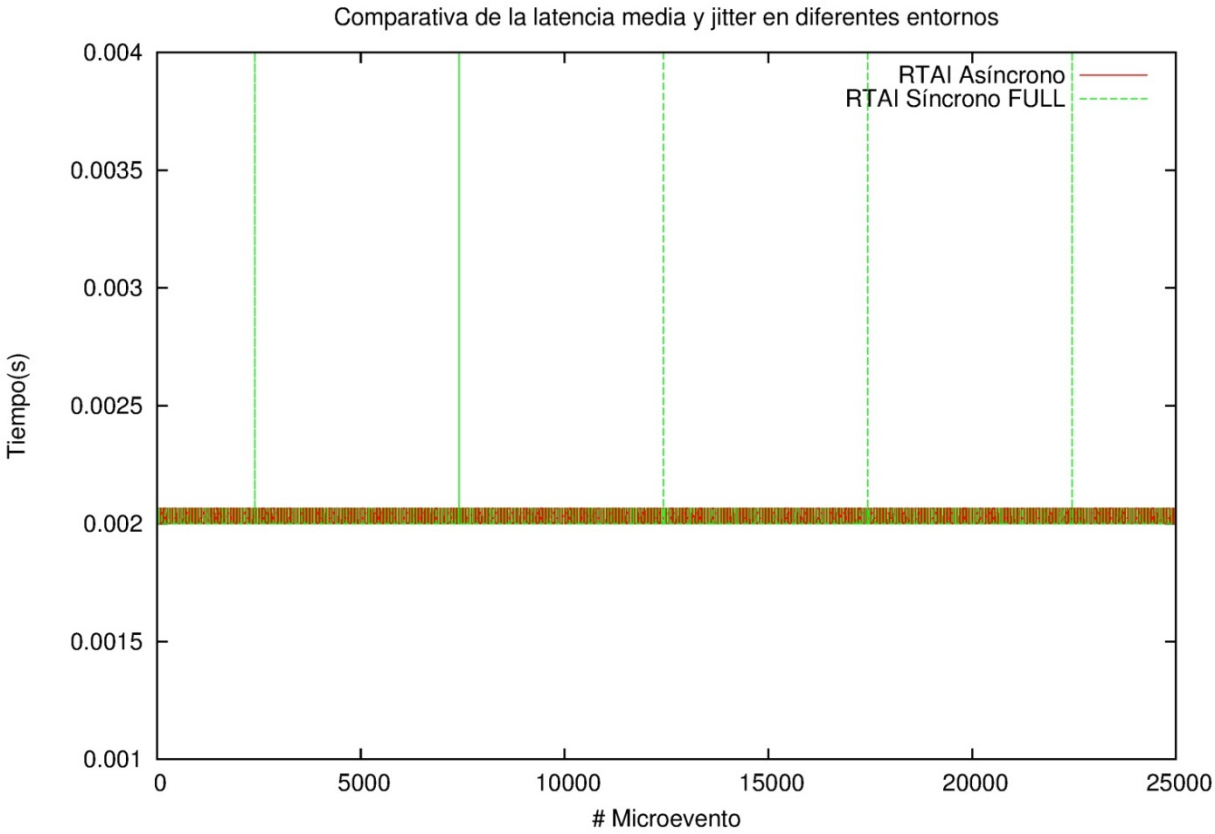


Figura 40: Ampliación de figura 39. Se observa la precisión de actuación del sistema en configuración asíncrona.

La única configuración posible que arroja resultados satisfactorios es la configuración asíncrona, en la que la tarjeta toma el control temporal del sistema, y de esta forma se rebaja el volumen de peticiones que se realizan sobre la DAQ por el PC. En modo síncrono realizamos peticiones que se encapsulan en comandos que se configuran por un modo de transferencia en forma de bulk, este tipo de peticiones controladas desde el PC colocan la transferencia de datos desde el PC a la DAQ en una situación crítica y necesaria para el buen funcionamiento del sistema. En cada momento de estimulación es necesario habilitar el canal USB para realizar la estimulación En la configuración asíncrona, en cambio, se encuentra un canal eficiente de comunicación entre el PC y la DAQ, la cual emite las muestras que recibe por una tubería habilitada durante el tiempo necesario para ese cometido.

En la estimulación síncrona observamos la pérdida de control de los tiempos de ejecución de las instrucciones síncronas para cualquier modo de configuración del protocolo USB, tanto high-speed como full-speed. La gestión de una tarea periódica continua en modo soft real time por parte del PC Pentium 4 uniprocesador elegido con sistema Linux y patch RTAI instalado no puede realizarse con unas garantías mínimas de latencia fija. La gestión

de las peticiones y bloqueo de las tareas continuo resulta inviable sobre este PC uniprocesador. Las tareas y procesos de control de Linux perturban la ejecución cíclica de alta precisión, y se introducen perdidas de control muy elevadas.

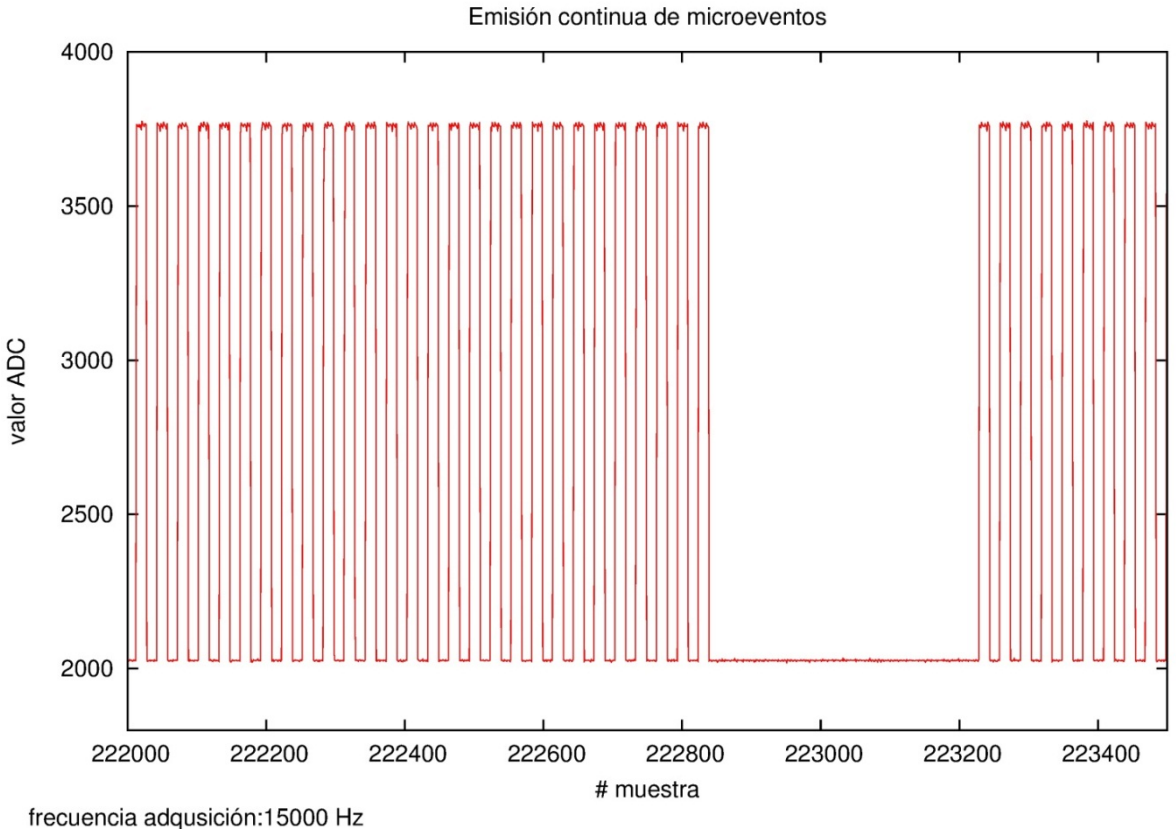


Figura 41: Fallo detectado en la estimulación síncrona. Esta pérdida de control registrada corresponde al macroevento que por segunda vez toma valor aproximado de 0.027 ms de la Figura 39.

Otra demostración más evidente de la ineficacia del control de tiempos por parte del sistema operativo Linux de propósito general y del sistema síncrono habilitado sobre RTAI instalado sobre el PC Pentium 4 (sistema con un solo procesador) es por medio de una estimulación o señal de control que requiera de mayor complejidad. El evento estimulador que se envía en este caso es un macroevento compuesto de 2 microeventos más un espacio de 1 milisegundo. Se trata de una prueba de emisión continua y periódica, de periodo 5 milisegundos. La ineficacia del control de los tiempos del sistema Linux y la demostración empírica de la necesidad de utilizar RTAI para obtener una precisión en la ejecución de tareas, quedan patentes en el análisis de los jitter que sufre la emisión continua de macroeventos.

5.2.2 Emisión continua de macroeventos

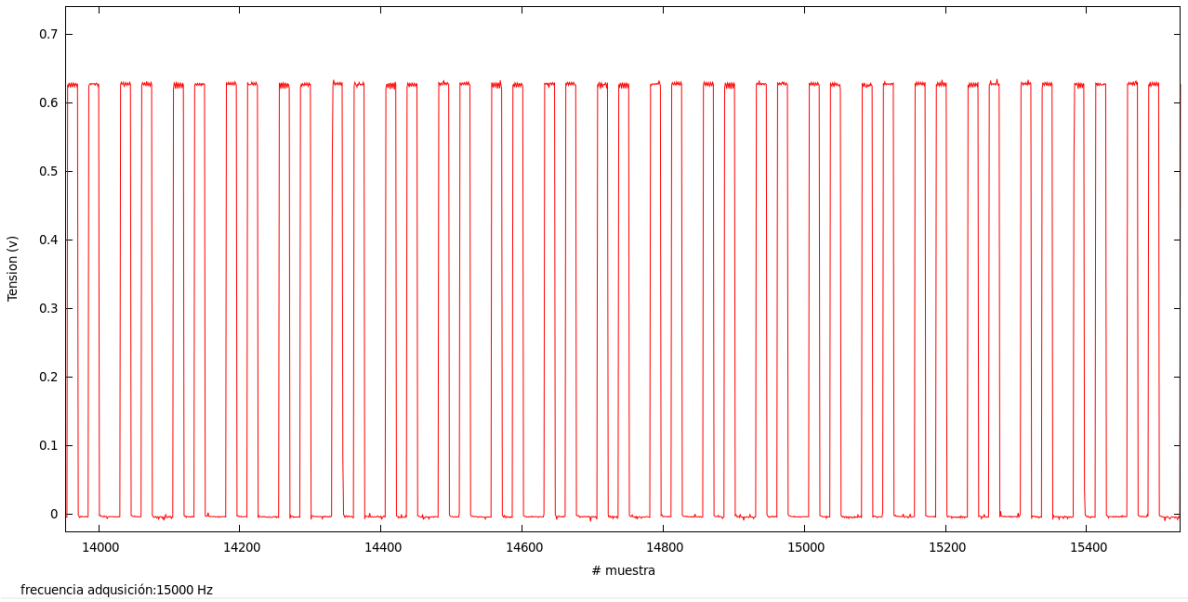


Figura 42: Ejemplo de macroevento de estimulación, grabación realizada por medio del programa cmd (vesase Anexo 1.4 Programa de adquisición asíncrona en modo usuario con comedilib). Con este programa en vez de Comedirecord el valor de tensión está expresado en voltios en vez de valor de conversor A/D como por ejemplo en las Figuras 33,34, 35…).

Se realiza la grabación y se realiza un análisis de la latencia de cada macroevento, se contabilizan los puntos obtenidos desde la detección de un microevento hasta la detección de dos más. El código de Octave es similar al empleado para el cálculo de la latencia de cada periodo de estimulación por microevento (véase Fig. 33).

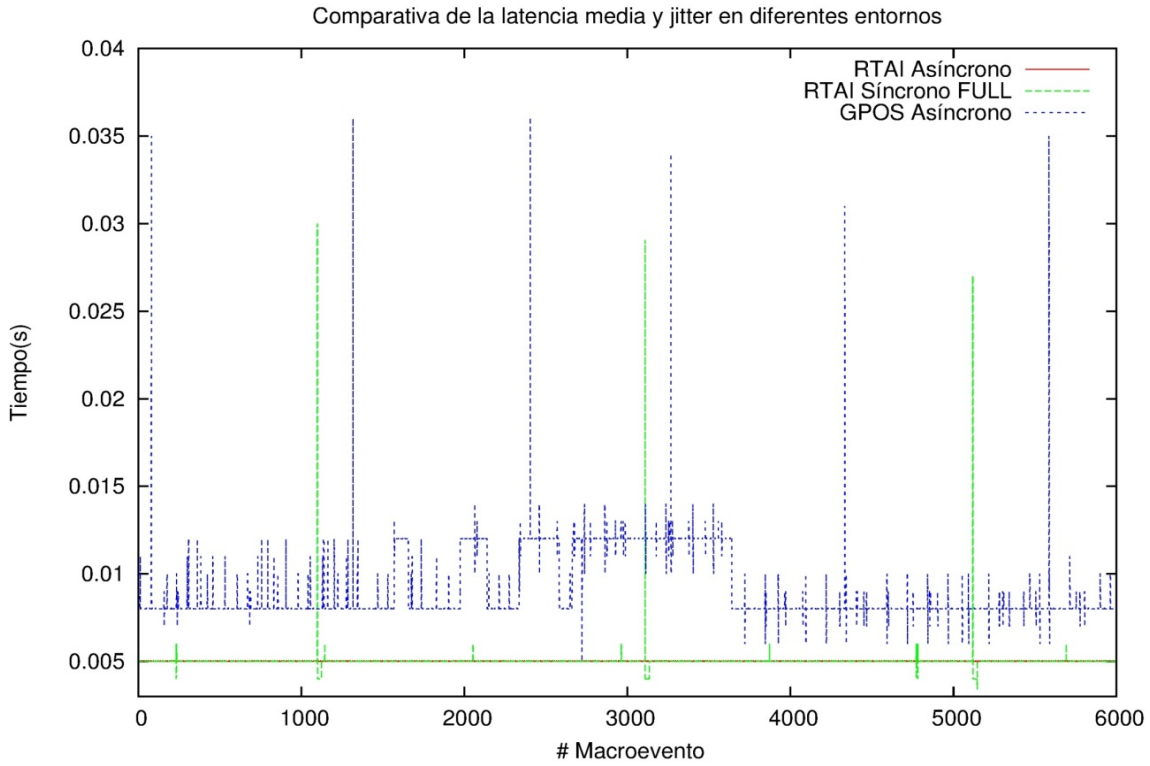


Figura 43: Jitter en la latencia de la emisión de un macroevento. El jitter se obtiene del análisis de las grabaciones de estimulación de alta precisión, en entornos de tiempo real por medio de funciones síncronas y asíncronas para el manejo de la tarjeta DAQ y en entornos de sistemas operativos de propósito general. El protocolo USB está en modo 2.0 full-speed en las configuraciones por medio de funciones síncronas y en high-speed para configuración asíncronas.

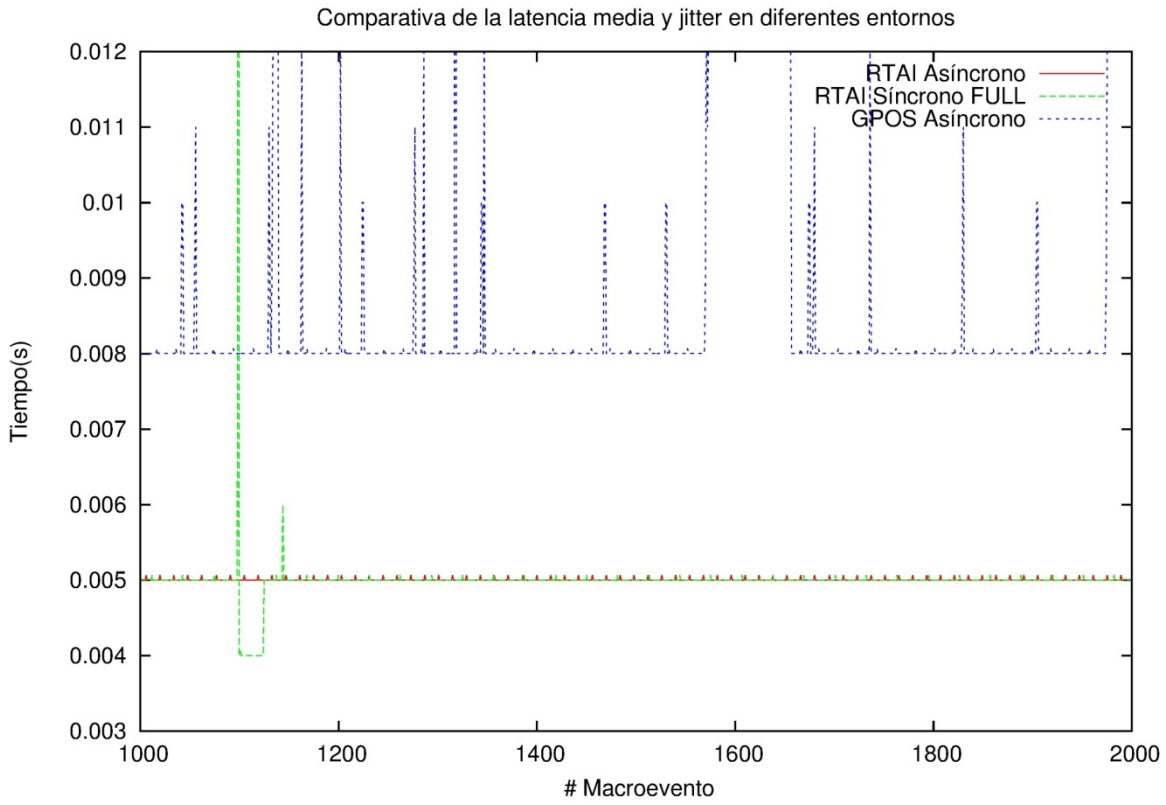


Figura 44: Ampliación de figura 44. A mayor detalle se observa que la configuración asíncrona sobre el PC Pentium 4 es la única configuración que no muestra jitter representativos en la latencia de la emisión de un macroevento de todas las pruebas de estimulación de alta precisión realizadas diversos en entornos.

La necesidad de utilizar RTAI se demuestra al analizar las características temporales de estimulación que se obtienen en el sistema síncrono sobre GPOS al observar la Figura 44. En esta configuración Linux es el encargado de la activación periódica de la tarea de emisión de señal, así como de la temporización del periodo. La línea de color azul de la Figura 44 dibuja los tiempos de ejecución de un macroeveto, donde se solicita realizar los estímulos cada 5 milisegundos. Se observa que la emisión de macroeventos en media requiere de un tiempo medio de 8 milisegundos e incluso hay tramos en los que se va a 16 milisegundos.

En la misma Figura 44 se observa que los sistemas de temporización de RTAI ofrecen temporización precisa de activación incluso en modo soft real time. En una configuración síncrona sobe el sistema RTAI se obtiene una emisión cuasi continua de macroeventos con

periodicidad la exigida excepto en algunos puntos en los que se cree que el control temporal del sistema es absorbido por alguna otra tarea de mantenimiento de Linux. También los escalones registrados en los tiempos medios de latencia de macroevento en configuración GPOS demuestran la inviabilidad de implementar un sistema de tiempo real sobre un uniprocesador en una configuración soft real time con RTAI. El sistema uniprocesador parece que no puede gestionar todo el sistema en conjunto, el sistema de emisión continúa con condiciones temporales más las demás tareas que realiza el sistema operativo Linux para su normal funcionamiento.

Se concluye que el sistema implementado en una configuración síncrona sobre un Pentium 4 a 3.2 GHz, sistema uniprocesador, con Linux y su patch de tiempo real RTAI no responde a un sistema de tiempo real con latencias conocidas y previsibles. El sistema operativo Linux con patch RTAI en modo soft real time, en un PC con un solo procesador, no ofrece buenos resultados en la gestión o control de un sistema de ejecución continua de peticiones síncronas USB con una precisión de milisegundos.

En la siguiente sección se realiza la prueba sobre un sistema de mayor número de procesadores, para demostrar que sobre un sistema que dedique más recursos a las tareas de RTAI es posible implementar un sistema real time soft de estimulación de alta precisión en modo síncrono a través de USB.

5.3 Estimulación de alta precisión en un sistema implementado en un PC con varios procesadores.

En esta sección se utiliza un PC quad-core i7 a 3.40 GHz (cuatro CPU's) con la misma versión de Linux 10.04 y con las mismos patch RTAI, librerías Comedi y módulos de las DAQ's USBDUX. En esta nueva configuración el planificador de RTAI obtiene un procesador dedicado para la ejecución de sus tareas mientras el sistema operativo Linux puede emplear cualquiera de los tres núcleos o procesadores restantes sin perturbar la ejecución cíclica de la tarea de estimulación. Los tarjetas DAQ's USBDUX son reconocidas y habilitadas para funcionar en USB 2.0 en modo High-speed, esto es debido a la implementación de su controlador que autoconfigura el protocolo USB a la máxima velocidad posible.

Los script de iniciación del sistema y los códigos empleados son idénticos en ambos sistemas. Lo único que se corrige es la forma de fijar la latencia mínima de un microevento para una configuración high-speed del protocolo USB debido a las conclusiones obtenidas por pruebas específicas realizadas que evaluaron otras formas de ajuste de la duración del pulso que se desea emitir. En este caso, se emplea la llamada recursiva de instrucciones síncronas de emisión simple hasta obtener la latencia de duración deseada, es decir, para conseguir 1 milisegundo de duración no se utilizará un rt_wait o rt_busy como en la implementación anterior, se llamara dos veces a la función comedi_data_read de latencia fija 500 µs.

Se comprueba la eficacia del sistema en la emisión de señales analógicas de alta precisión en diferentes entornos GPOS o RTAI en configuración síncrona o asíncrona. De nuevo se analizan los tiempos de latencia y jitter que ocurren en la duración y en la periodicidad de la emisión de microeventos o macroeventos similares a los de la experimentación expuesta el capítulo anterior 5.2.

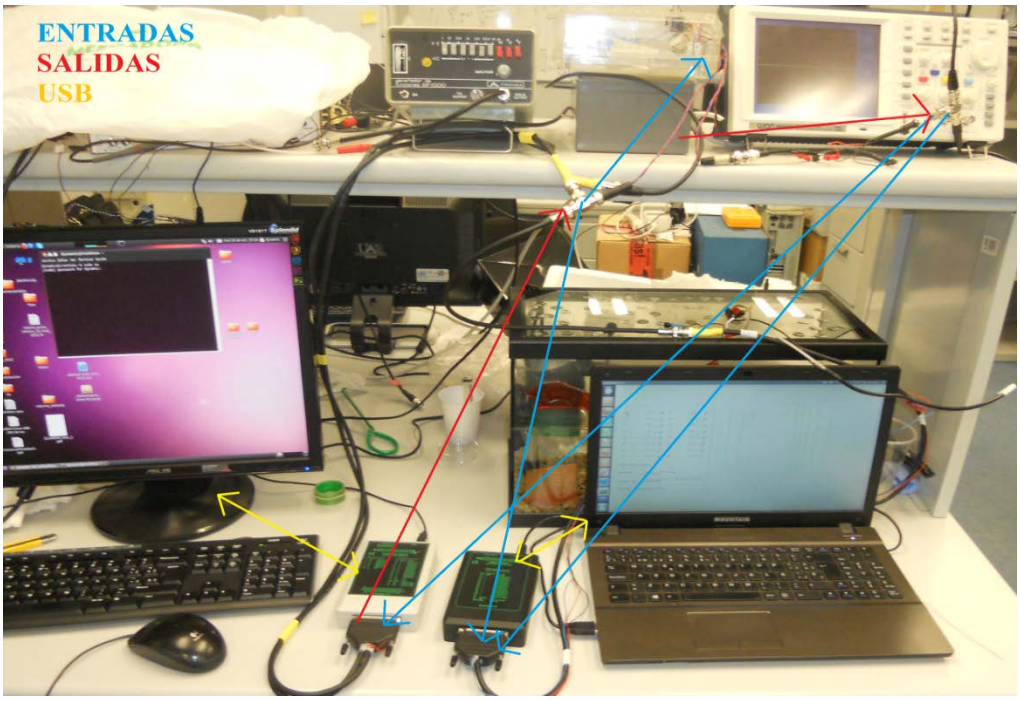


Figura 45: integración del sistema en el PC con varios procesadores.

5.3.1 Emisión continua de microeventos

5.3.1.1 Comparativa de precisión en duración: Latencia media y jitter

El análisis de latencia muestra que los sistemas en GPOS dado el nivel de procesado que ofrece esta nueva implementación sobre el PC de varios procesadores y mayores prestaciones se obtiene una estimulación de precisión de milisegundos. En configuración high-speed el protocolo USB 2.0 en un entorno RTAI, en una configuración síncrona, ofrece unas latencias fijas y precisas en la duración de la señal de valor de tensión en high que construye los micreventos que se utilizan para implementar un sistema de emisión de alta precisión temporal. El uso de las instrucciones síncronas en un sistema con un procesador dedicado únicamente a realizar la emisión continua de señal analógica o tareas de manipulación de las DAQ's ofrece muy buenos resultados a la vista del estudio de jitter las grabaciones realizadas (véase Fig. 46). La técnica de composición del impulso por medio de las instrucciones síncronas de Comedi permite la emisión de pulsos múltiplos en duración del pulso de emisión mínima de duración 500 µs. Las latencias medias y los jitters que sufre el sistema en una configuración asíncrona son similares a las obtenidas en la integración anterior, sobre el PC uniprocesador. En esta configuración la DAQ USBDUX-sigma impone la temporización del sistema.

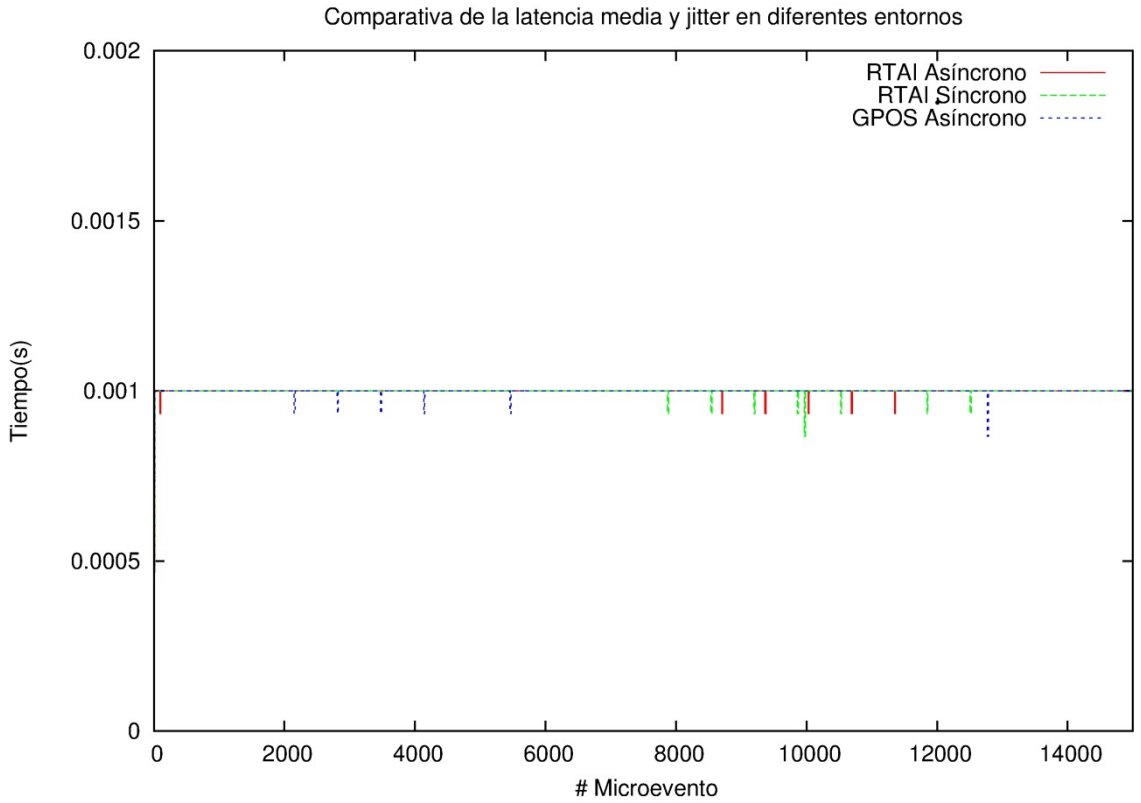


Figura 46: Gráfica comparativa de los jitter registrados en la duración del valor de tensión high, parte de señal, que compone un microevento.

5.3.1.2 Comparativa de precisión en momento de actuación: Latencia media y jitter

En este capítulo, se realiza el estudio de la latencia media en la periodicidad de emisión de microeventos y así determinar la precisión en momento de actuación, la latencia media que se espera registrar es de 2 milisegundos, periodo de un microevento (véase Fig. 31).

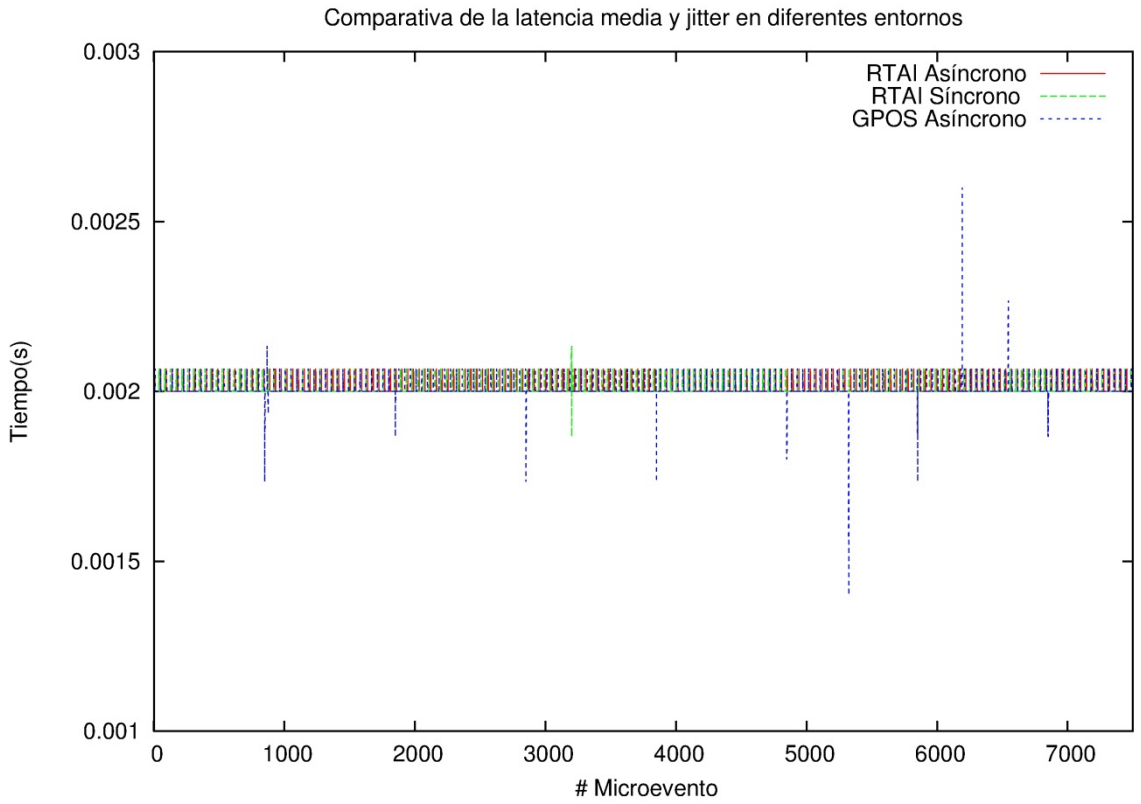


Figura 47: Jitter en la latencia de la periodicidad de emisión de un microevento, con esta grafica se analiza la precisión de actuación del ciclo estimulador, la prueba de emisión se lleva a cabo en entornos de tiempo real por medio de funciones síncronas y asíncronas para el manejo de la tarjeta DAQ y en entornos de sistemas operativos de propósito general.

En los sistemas GPOS la temporización mejora sustancialmente respecto al sistema uniprocesador anterior véase figura 37. En la configuración asíncrona es la DAQ la que impone la temporización d alta precisión.

La configuración síncrona del sistema ofrece una precisión muy buena tanto en duración como en momento de actuación. En un sistema con varios procesadores con un sistema Linux con patch RTAI, las instrucciones síncronas de Comedi de emisión/adquisición de una muestra simple y la DAQ USBDUX-sigma componen un sistema que puede ser empleado para la emisión de alta precisión. Esto es necesario para la implementación de ciclos cerrados de estimulación dependiente de la actividad.

5.3.2 Emisión continua de macroeventos

Se realiza la misma prueba en los tres entornos analizados anteriormente, se analiza la precisión de actuación del sistema de emisión, en la emisión continua de macroeventos idénticos a los de la figura 42.

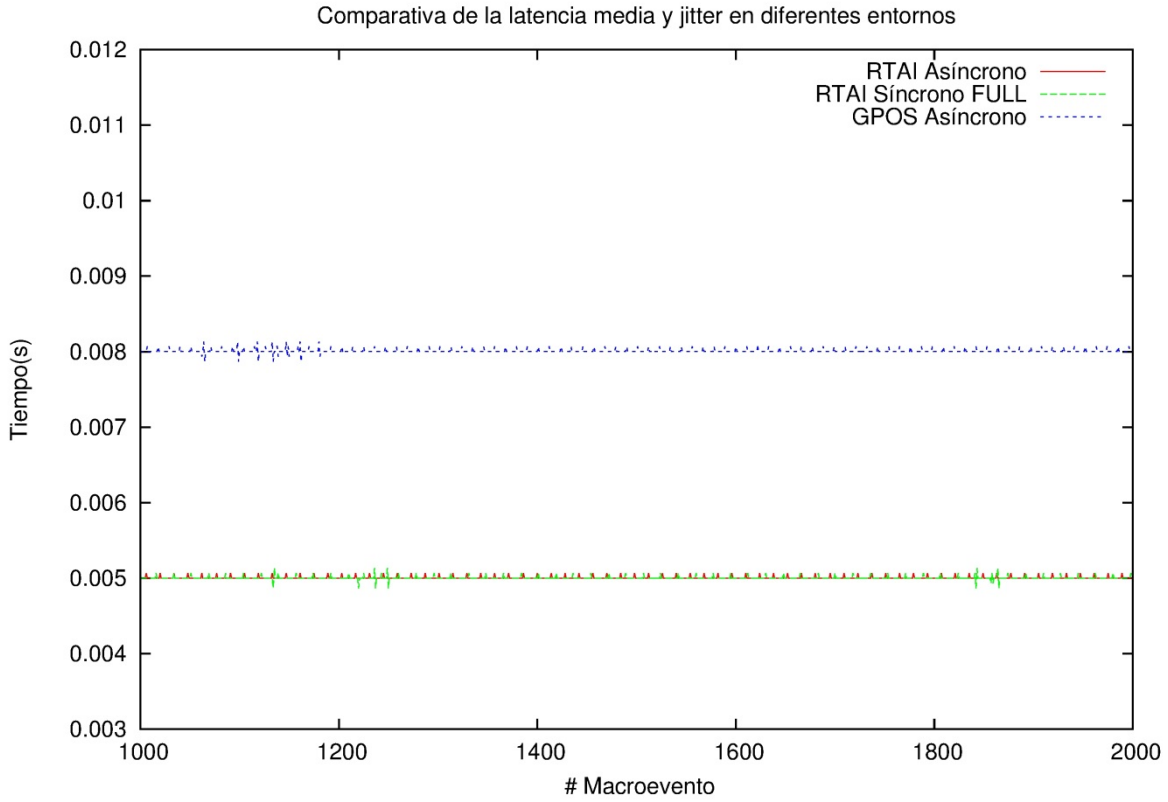


Figura 48: Jitter en la latencia de la emisión de un macroevento. El jitter se obtiene del análisis de las grabaciones de estimulación de alta precisión, en entornos de tiempo real por medio de funciones síncronas y asíncronas para el manejo de la tarjeta DAQ y en entornos de sistemas operativos de propósito general. Se observa la potencia de procesado del PC i7 con RTAI.

A la vista de los resultados obtenidos, la implementación del sistema de tiempo real, obtenido por la instalación del patch RTAI sobre un Linux kernel genérico, que trabaja en configuración soft realt time (necesidad impuesta por el USB) debe realizarse en un PC con varios procesadores. La utilización de un procesador para la ejecución de las tareas RTAI y de otro o más procesadores para la ejecución de las tareas de gestión de Linux posibilita la creación de un verdadero sistema de emisión/adquisición de alta precisión. En la figura 43, se observan las pérdidas de control de las tareas RTAI para la emisión continua de macroeventos. De la misma figura 43, el caso del entorno GPOS es más revelador, se impone una periodicidad de 5 ms al proceso de emisión, el sistema ofrece de media 8 ms pero en momentos esporádicos la periodicidad ofrecida en media es de 16 ms, se cree que el nivel de procesado que ofrece un solo procesador que ejecuta de manera concurrente la tarea de RTAI y los procesos Linux no es suficiente y es probable que algún proceso de Linux interrumpe al proceso de emisión.

5.4 Ciclo cerrado de estimulación dependiente de la actividad

Los ciclos cerrados que interactuaran con la neurona artificial requieren de un control temporal de milisegundos. La única implementación que arroja resultados esperanzadores para la creación de ciclos cerrados de estimulación de alta precisión, es sobre un PC con varios procesadores que permita la ejecución de las tareas en modo soft real time asegurando unas latencias fijas. Por tanto, en esta sección, las pruebas de ciclo cerrado se realizan sobre el PC quad-core i7 a 3.40 GHz con la misma versión de Linux 10.04 y con las mismos patch RTAI, librerías Comedi y módulos de las DAQ's USBDUX que anteriormente.

Se analizan dos implementaciones posibles, una a través de funciones síncronas de Comedi para la adquisición/emisión de una muestra, configuración llamada síncrona; y en la otra implementación se utilizan la potencia de adquisición que ofrece las DAQ's en modo asíncrono y las funciones de señalización y adquisición de kcomedlib que posibilitan la manipulación de una muestra simple en una configuración isócrona del protocolo USB. En la adquisición asíncrona la DAQ impone la temporización del sistema caracterizada por la tasa de adquisición. Se aprovechan los espacios temporales que ocurren entre la detección por Comedi de la llegada de una nueva muestra adquirida y la anterior, para implementar el algoritmo de detección de eventos y un sistema de emisión de alta precisión síncrona por medio de las funciones síncronas de Comedi la estimulación.

5.4.1 Configuración síncrona

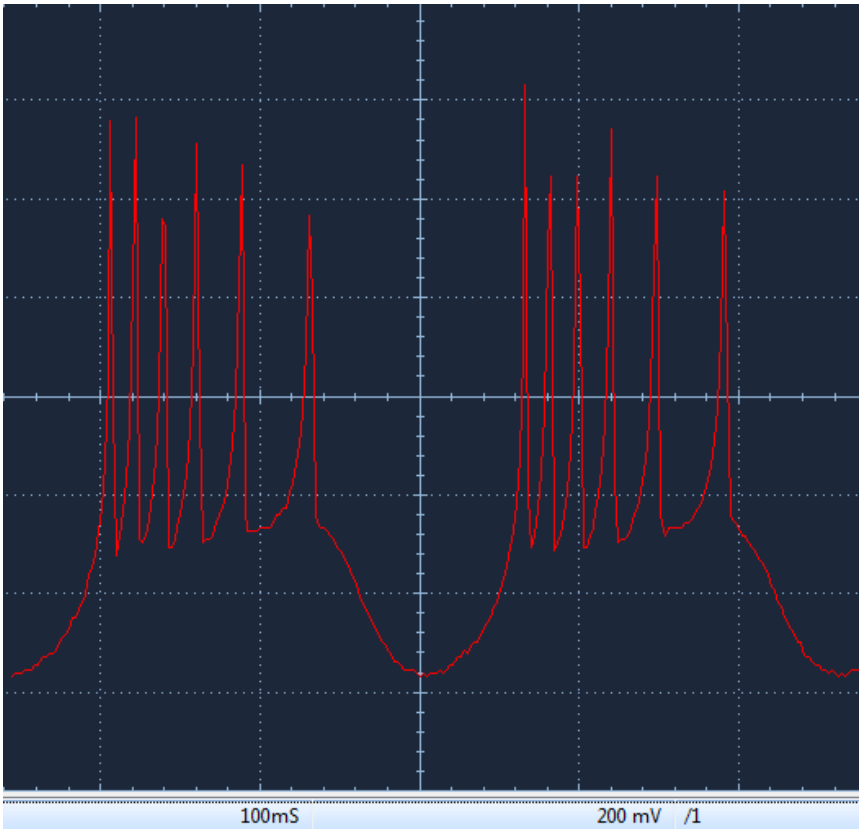


Figura 49: Captura Osciloscopio del estado de excitación de la neurona artificial. Gracias al osciloscopio se analizan las características de la señal de salida de la neurona artificial y

así se determinan las necesidades del muestreo de la señal para mantener un nivel de fidelidad entre la señal muestreada y la señal original.

En este estado concreto de excitación, la señal de salida de la neurona se muestra con una forma de una actividad en ráfagas o *bursts*. La monitorización de la señal necesita percibir los grupos de spikes. La escala temporal de la gráfica es de 100 ms por lo que los mínimos locales del potencial en las zonas depolarizadas, que determinan la sucesión de dos bursts podrían ser caracterizados como una señal sinusoidal de frecuencia 20 Hz. En cambio si se tiene como objetivo la detección del máximo que determina un spike, es necesaria la monitorización a una tasa de al menos 50 Hz por lo que se podría realizar la tarea de monitorización como máximo cada 20 ms según el teorema de Nyquist. Observados los resultados obtenidos sobre el generador de señales (figura 50), es necesario submuestrear el periodo de la señal al menos con 4 puntos para obtener una grabación fiel de la señal.

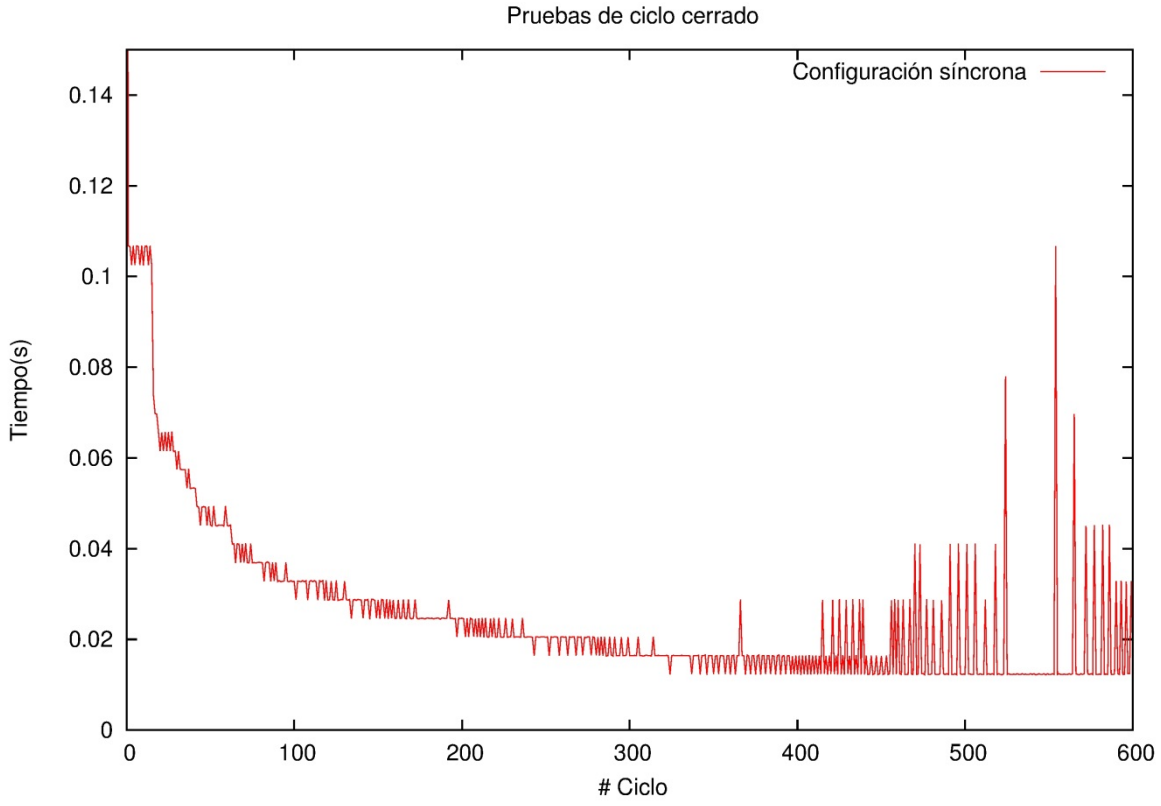


Figura 50: Jitter en la latencia media de la ejecución de los ciclos cerrados dependientes de la actividad que tienen como objetivo la detección de mínimos en una señal sinusoidal de frecuencia variable (se varía la frecuencia de 10 Hz a 100 Hz) que emite el generador de funciones. Los ciclos se implementan en una configuración síncrona con un periodo de ciclo de 4 milisegundos. La figura 51 recoge la grabación realizada por la USBDUX-fast del sistema.

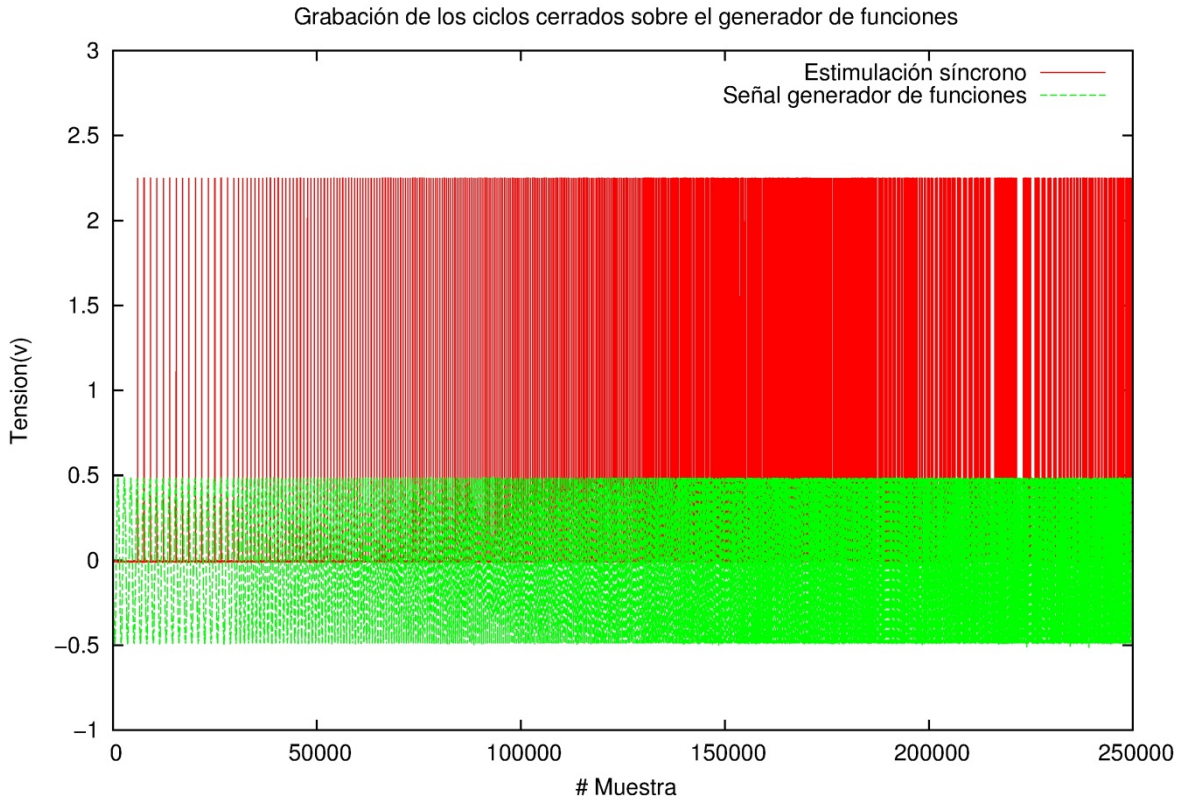


Figura 51: Grabación de la prueba que determina la viabilidad y límites de los ciclos cerrados en una configuración síncrona.

Como podemos observar en la figura 50, a pesar de que la periodicidad de los ciclos ofrece posibilidades teóricas de muestreo de señales de hasta 125 Hz (ciclos de periodo de 4 milisegundos), no es posible implementar ciclos cerrados sobre funciones con variaciones de frecuencias cercanas a la tasa de Nyquist de nuestro sistema periódico. La tasa de Nyquist no es garantía de posibilidad de muestreo fiable de la señal, los tiempos de ejecución de las tareas de adquisición no son estrictos sino que se ejecutan en soft real time. Es necesario realizar un submuestreo de la señal de al menos 4 puntos de la variación mínima que se desee detectar.

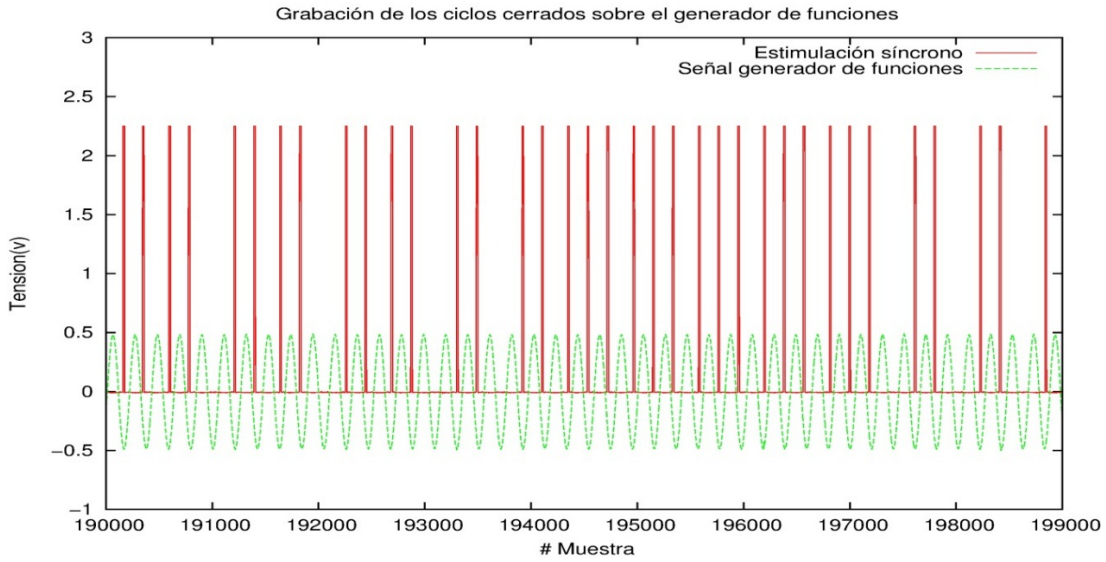


Figura 52: Grabación sobre la inviabilidad del sistema: La frecuencia de la señal seno es de 90 Hz. A pesar de la supuesta tasa de muestreo que ofrecen los ciclos de 250 Hz no es posible monitorizar correctamente frecuencias cercanas a la tasa de Nyquist.

La neurona artificial puede ser monitorizada con un nivel de fidelidad alto. Con la configuración del sistema en ciclos de 4 0 5 milisegundos podemos obtener frecuencias de 250 o 200 Hz de tasa de muestreo aun que se ha demostrado que en la práctica se ve reducida a 60 Hz. Observando las grabaciones de la neurona artificial (figura 7, figura 8 o figura 49), las frecuencias son suficientes para manipular la salida de la neurona artificial.

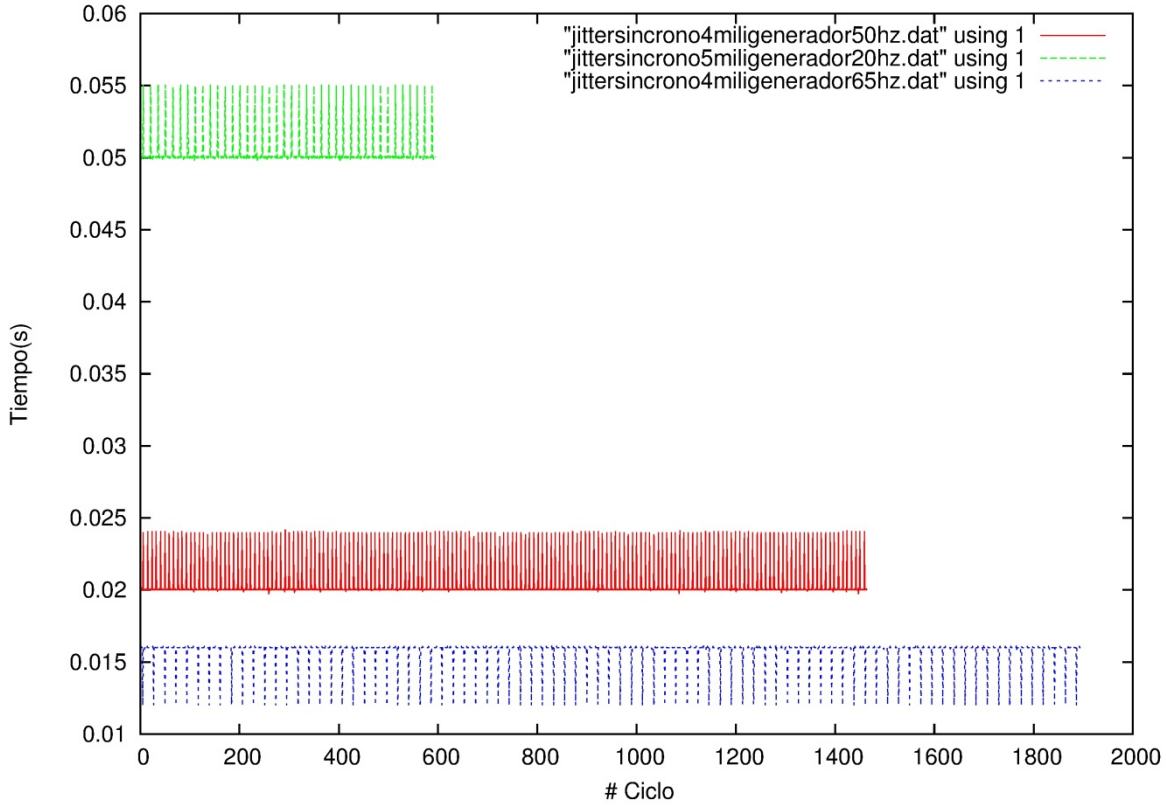


Figura 53: Jitter y latencias medias de las grabaciones de 30 ms de duración realizadas en un configuración síncrona. En la leyenda de la figura se expresa la prueba realizada. Por ejemplo en verde, en ciclos de 4 milisegundos se realiza la emisión de microevento cada mínimo detectado en la función sinusoidal de 20 Hz.

Gracias a las conclusiones obtenidas sobre el generador de señales se realizan los ciclos cerrados sobre la neurona artificial. En estas pruebas se empleará la configuración síncrona con periodo de ciclo 4 milisegundos, con microeventos de periodo 2 ms como señal de estimulación que se emite al detectar el evento objetivo de análisis. El evento objeto de análisis consiste en detectar en la señal un número concreto de mínimos o máximos en una ventana de tiempo determinada.

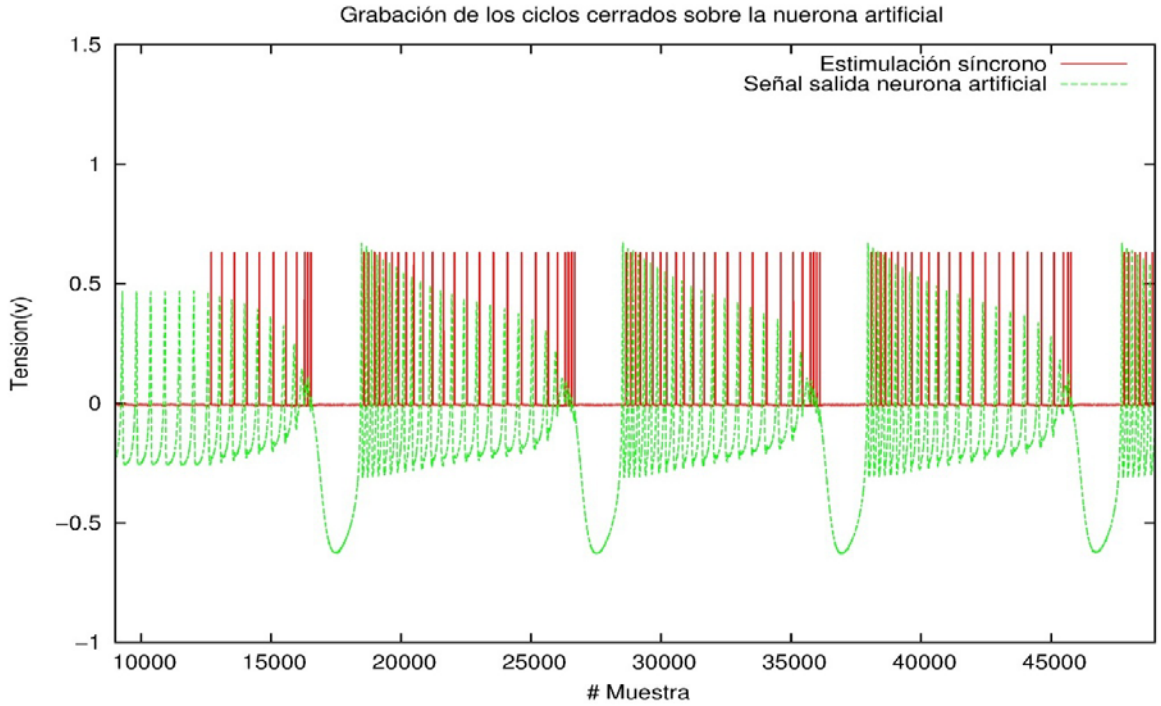


Figura 54: Ejemplo de ciclo cerrado actuando sobre la neurona artificial, Se muestra un tramo de la grabación. Se corresponde al inicio de los ciclos cerrados sobre la neurona. La detección de un spike, genera la emisión de un microevento que provoca el cambio en el estado de excitación de la neurona artificial.

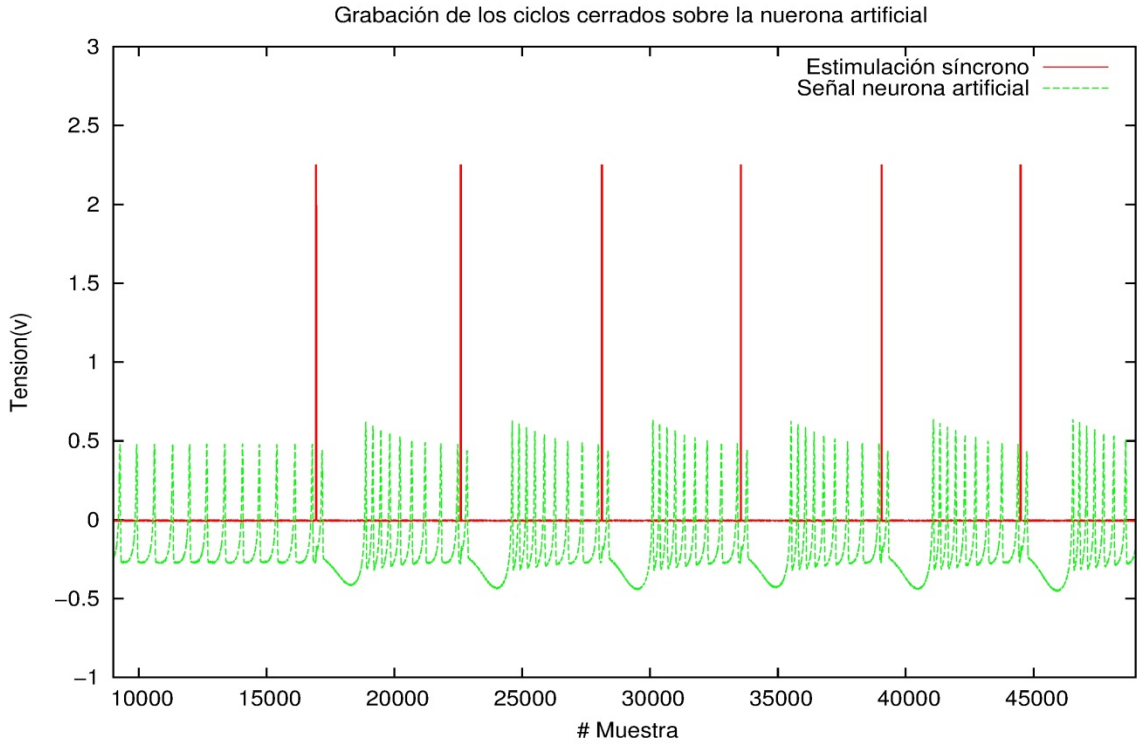


Figura 55: Nuevo objetivo del ciclo cerrado actuando sobre la neurona artificial. Se produce la estimulación cada 10 spikes, esta vez detectados por medio de los mínimos que se dan entre las subidas de tensión que determinan los spikes. La detección de objetivo, provoca la emisión de un microevento que estimula el cambio en el estado de excitación de la neurona artificial.

5.4.2 Configuración mixta:

Se realizan pruebas empleado una configuración asíncrona de la DAQ para la adquisición de datos y volcado al PC de manera controlada por las funciones de señalización de RTAI y de manipulación de una única muestra de Kcomedlib. Entre cada recepción de una nueva muestra se realiza la tarea o algoritmo de detección de objetivo, el objetivo se fija en la detección de máximos en una señal sinusoidal de frecuencia 100 Hz emitida por el generador de funciones, y la emisión de señal de analógica. La estimulación o emisión de señal se realiza por medio de las instrucciones de emisión síncronas de Comedi. El estímulo consiste en un pulso de duración 500 μs. Se realizan pruebas con la tasa de muestreo superiores a las alcanzables por la configuración síncrona. Se analiza la viabilidad para la realización de ciclos cerrados de latencia media de 1 ms (configuración del comando para adquisición asíncrona a 1 kHz) o a 2 ms (a 500 Hz).

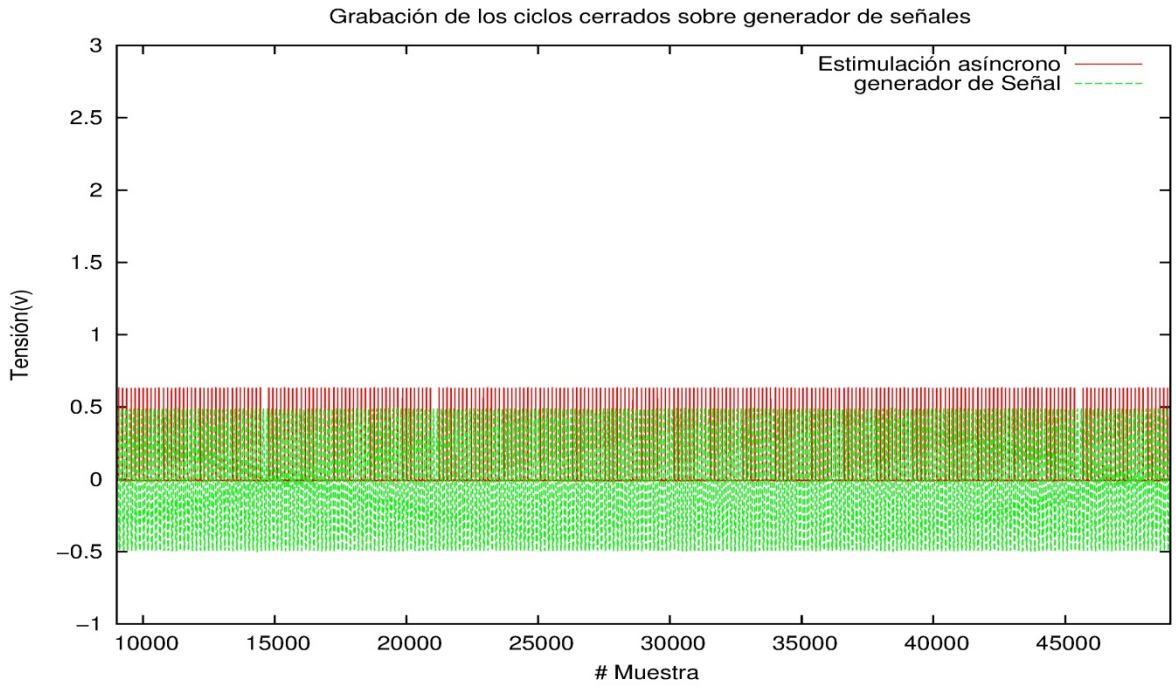


Figura 56: Grabación de las pruebas de ciclos cerrados de estimulación gobernados por el objetivo de detección de máximos en una señal sinusoidal de frecuencia 100 Hz en una configuración mixta del sistema.

Se puede observar en la figura 56 que se producen fallos en la emisión de señal en determinados puntos de la grabación (por ejemplo entorno # muestras 22000 se observa la señal del generador en verde claramente). Se realiza un estudio del jitter y latencia media entre cada estimulo emitido desde la DAQ USBDUX-sigma.

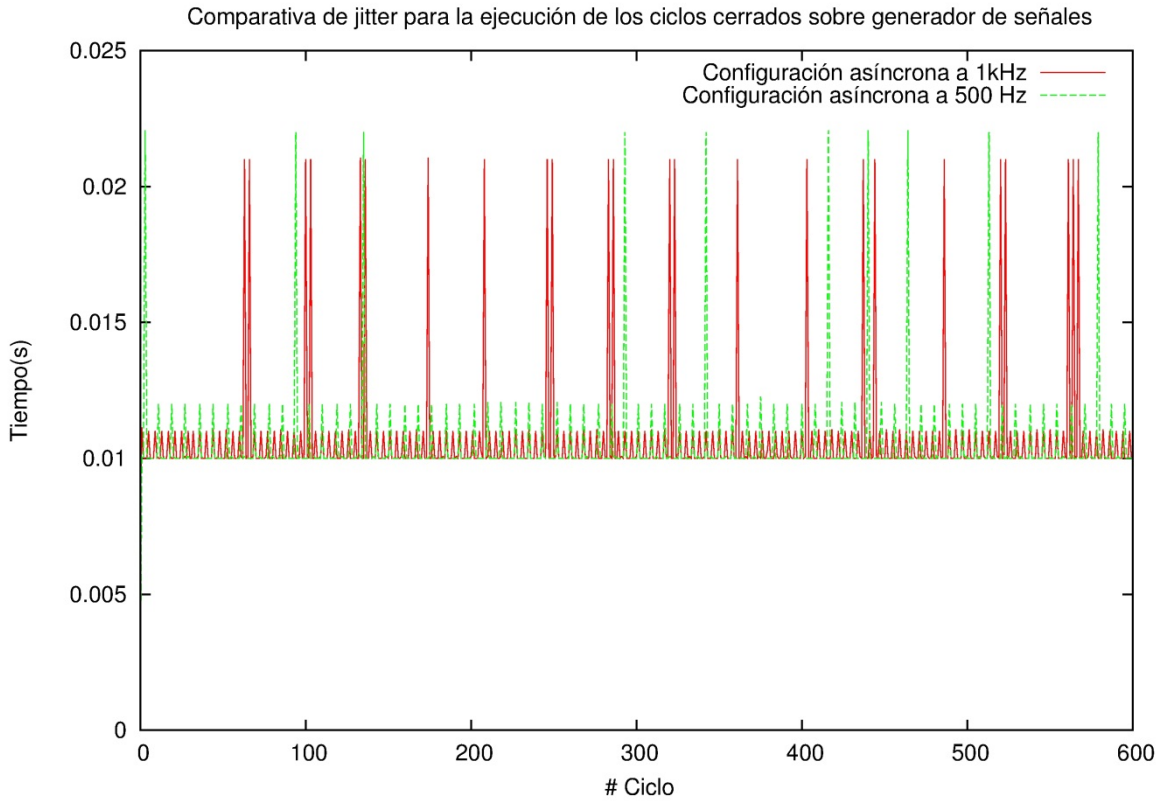


Figura 57: Test de latencia media y jitter que sufre el sistema de ciclos cerrados en configuración mixta.

El sistema en configuración mixta ofrece peores resultados que en configuración síncrona (véase figura 53). La señal sinusoidal de 100 Hz provoca la emisión de estímulos cada 10 ms. Los ciclos de duración 20 ms corresponden a errores de funcionamiento del sistema ejecutor de ciclos cerrados. El sistema en esta configuración es incapaz de ofrecer un sistema con latencia fija en media. La ausencia de emisión del impulso puede deberse a la etapa de adquisición o a la propia ejecución de las instrucciones síncronas de emisión de señal. Las pruebas realizadas en la adquisición asíncrona de señal arrojan resultados muy positivos al igual que las de estimulación de alta precisión por esta razón se implementa los ciclos cerrados en esta configuración mixta pero a la vista de los resultados no resulta una configuración que ofrezca mejoras respecto la configuración síncrona.

5.4.1 Configuración asíncrona

La etapa de adquisición por medio de un comando Comedi se configura para trabajar a una tasa de muestreo de 1 KHz (máximo posible según fabricante). El algoritmo de detección detecta máximos en sobre una señal de 100 Hz generada desde el generador de funciones. La emisión se configura asíncrona para realizar secuencias periódicas de emisión de muestras cada milisegundo (también se configura a 1 kHz), en caso de detectarse objetivo se sirve a la DAQ desde el Pc a través de la tubería isócrona que habilita el protocolo USB una muestra de valor 0.65 V aproximadamente. Al igual que en las otras pruebas se evalúa el jitter que sufre el sistema de ciclo cerrado soft real time.

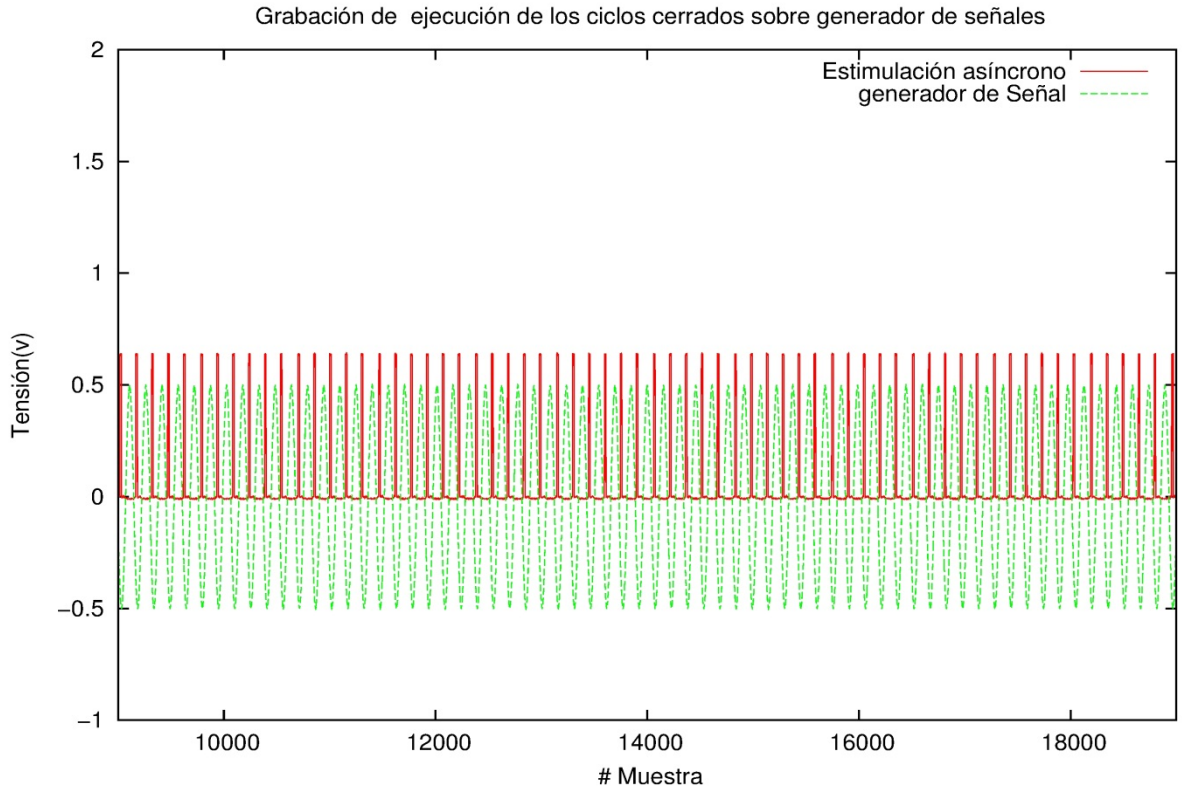


Figura 58: grabación de los ciclos cerrados sobre generador de señales en una configuración asíncrona total del sistema, tanto etapa de adquisición como etapa de estimulación. La implementación resulta viable para la monitorización eficiente de la señal de 100 Hz que emite el generador de señales, cada máximo se emite un estímulo.

Realizando el estudio del jitter observamos la potencia de la implementación de la configuración asíncrona ofrece tanto en adquisición como en la emisión. La emisión ocurre cada 10 ms (análisis realizado sobre señal de 100 Hz). También hay que tener en cuenta que pueden ocurrir fallos aunque en este ejemplo no se hayan detectado. Si ocurren estos fallos no serían detectados. El sistema en configuración asíncrona es otra solución para la implementación de ciclos cerrados en soft real time sobre la neurona artificial.

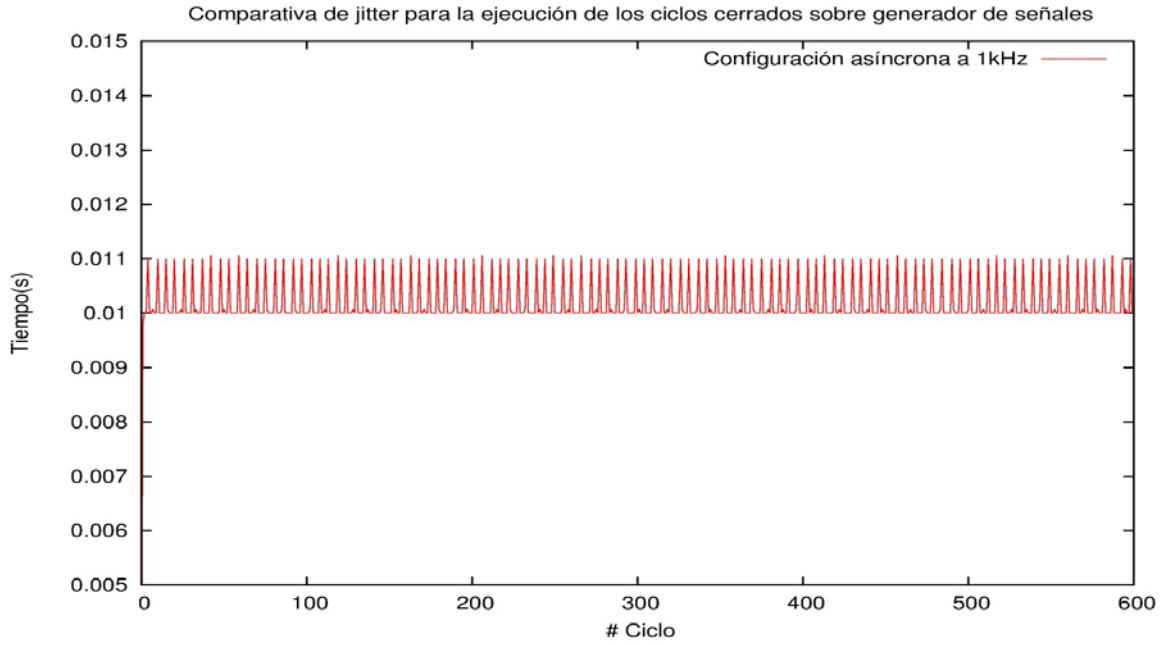


Figura 59 : test de jitter de la señal emitida por la USBDUX-sigma en la realización de ciclos cerrados en una configuración asíncrona de emisión y recepción. El error del sistema es de 1 milisegundo.

Sobre la neurona artificial se configura el algoritmo en la detección de mínimos en el interior de la señal en estado o modo burst. La precisión que ofrecen los conversores A/D y la tasa de muestreo de 1 kHz permiten detectar en tiempo real la pequeña perturbación que produce la señal de estimulación que se envía desde la USBDUX-sigma cada mínimo en el burts registrado.

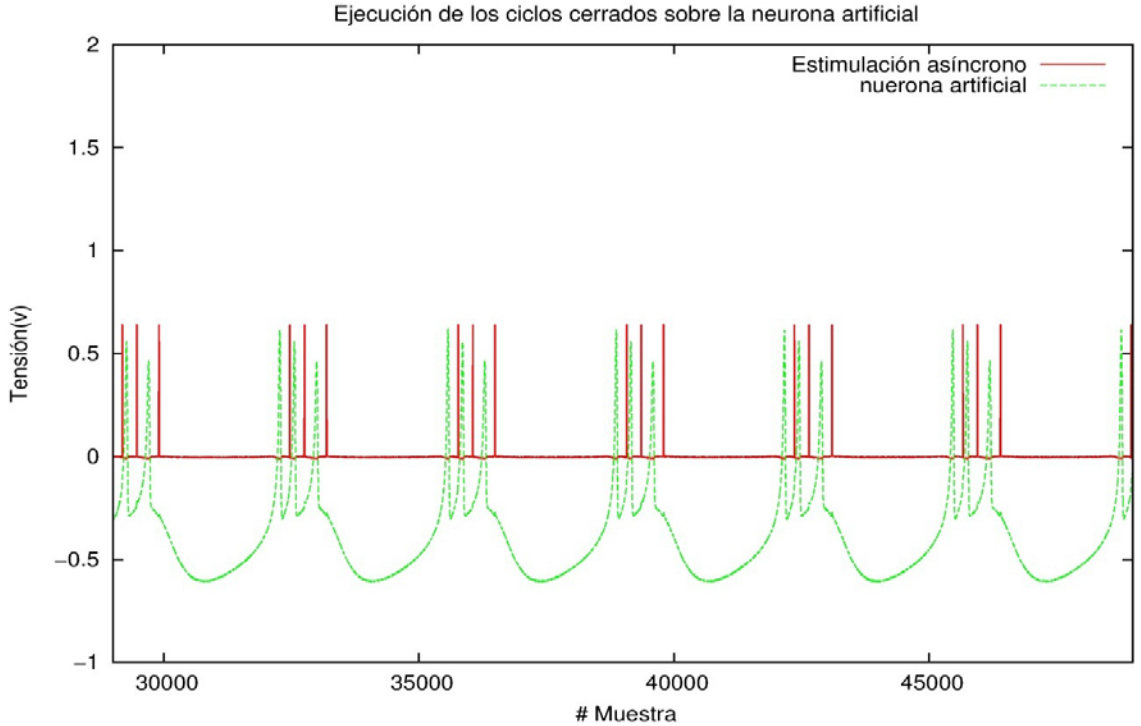


Figura 60: ejemplo de ciclos cerrados implementados en configuración asíncrona interactuando sobre la neurona artificial. Se observa la eficacia incluso en la detección de mínimos.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Las DAQ's de la familia USBDUX a pesar de su reducido precio ofrecen buenas prestaciones tanto en la adquisición como en la emisión de señal. La DAQ USBDUX-fast y USBDUX-sigma son buenas herramientas para realizar el registro de señalización a tasas de muestreo configurables por medio de los comandos de Comedi. La USBDUX-sigma, a pesar de tener menor rango de frecuencias de adquisición, posee unos conversores analógicos digitales muy potentes (24 bits) y permite la emisión controlada de señal analógica. La precisión de las tarjetas en su configuración asíncrona es muy alta.

En las librerías Comedi encontramos una gran cantidad de funciones que permiten el uso eficiente de la DAQ por parte del sistema operativo Linux. Este proyecto que contiene controladores, librerías y herramientas para el manejo de device es totalmente gratuito. El controlador ofrecido por Comedi gobierna la DAQ's de manera correcta. En espacio de usuario o en sistemas de tiempo real es posible el manejo de las DAQ's.

El protocolo USB en tiempo real en la configuración de nuestro sistema no posee propiedades ni garantías de hard real time. El protocolo USB tiene cuatro modos de transferencia de datos. En este proyecto se analizan el modo bulk de transferencia y el modo isócrono. Es en este último modo donde podemos encontrar latencias fijas de llegada de los datos al PC desde la DAQ, pero sin control de corrección de errores. En esta configuración se realiza un sistema con garantías de soft real time demostradas en este proyecto como la estimulación y la adquisición realizada con la USBDUX-sigma. En el modo bulk, modo utilizado en la configuración síncrona del sistema (ya que las instrucciones síncronas de Comedi en el controlador de la USBDUX están implementadas de esta forma), se realiza la retransmisión de la función que manipula la DAQ en el caso de producirse un error pero carece de latencia fija de transmisión y el momento de actuación sobre la DAQ está impuesto por el sistema operativo que controla el sistema.

Para poder dotar de cierta precisión al sistema es necesario la instalación del patch RTAI sobre el sistema operativo Ubuntu (Linux). Con esta decisión de uso del patch RTAI, se obtiene un mecanismo que ofrece garantías de tiempo real y a su vez la posibilidad de utilización de todos los recursos que ofrece el sistema de propósito general en espacio usuario. La precisión temporal que ofrece este patch en la ejecución de tarea RTAI se convierte en soporte de control necesario para implementación de sistemas de precisión. La prioridad de las tareas en el uso de los recursos del sistema así como la precisión en la duración o momento de actuación son logros obtenidos gracias al uso de un sistema operativo en tiempo real.

En este trabajo se ha demostrado la necesidad de emplear sistemas dedicados a la hora de implementar sistemas en tiempo real soft con las DAQ's USBDUX. En la primera implementación del sistema se utiliza un PC con un solo procesador. Este procesador debe realizar las tareas de RTAI requeridas y a su vez dar soporte al sistema operativo de propósito general. La ejecución de tareas periódicas con alta precisión temporal resulta inviable en el sistema de soft real time implementado sobre el PC uniprocesador. Se producen pérdidas del control de la ejecución requerida. La implementación del sistema

sobre un PC con mayores prestaciones y varios procesadores posibilita la creación de un sistema soft real time que realice adquisición/emisión de señal analógica a través de USB.

Para aplicaciones open-loop, aplicaciones en las que el flujo de información se produce en un único sentido (por ejemplo el registro de señalización biológica en tiempo real soft o sistemas de control de motores que requieran de emisión de estimulación en precisión de milisegundos), el sistema implementado con el PC de varios procesadores con Linux y el patch RTAI y la DAQ USBDUX-sigma controlada por Comedi resulta viable en las dos configuraciones analizadas en este proyecto: tanto en la configuración síncrona a través de instrucciones de manipulación de una única muestra donde la temporización la impone el sistema de tiempo de RTAI, como en la configuración asíncrona, configurable a través de los command de Comedi, con temporización impuesta por la tarjeta DAQ y que utiliza las funciones de Kcomedilib, para dar sincronismo al sistema.

En los sistemas de ciclo cerrado, la complejidad del sistema y la necesidad de un control riguroso de los tiempos del sistema crecen. Al no poder ejecutar el sistema en un modo hard real time, modo en el que se aseguran las características temporales impuestas sobre la tarea, no es posible obtener un sistema con una precisión temporal alta. Las pruebas de ciclo cerrado realizadas en modo síncrono habilitan un sistema de precisión con un error igual a la duración del ciclo. Aun demostrada la viabilidad del sistema para la emisión/adquisición de alta precisión, la implementación de los ciclos cerrados con alta precisión temporal en este sistema parece inviable. La implementación del controlador de las DAQ's emplea un modo de transferencia en bulk en las instrucciones síncronas de muestra analógica simple, modo de transferencia no adecuado para sistemas en tiempo real pues no asegura el tiempo de transferencia de los datos del periférico al PC. En la configuración asíncrona de los ciclos cerrados se emplea el modo de transferencia adecuado para el desarrollo de sistemas en tiempo real soft (debido a que no hay control de errores se puede producir la pérdida de un dato), que es el modo isócrono de transferencia USB pero no se obtiene un comportamiento estricto temporal. El control temporal que se requiere para la ejecución de ciclos cerrados en esta implementación no puede ser asegurado sin que el PC host de USB pueda ejecutar funcionalidad en hard real time. A pesar de no obtenerse un sistema de tiempo real con características hard, las USBDUX resultan adecuadas para implementar un sistema soft real time de ciclos cerrados que trabaja con un error de 1 milisegundo en su configuración de máxima frecuencia de 1 kHz.

Se concluye por tanto que en aplicaciones en las que el flujo sea en una dirección, ciclos abiertos, debido a la potencia de procesado, tiempos del sistema y velocidad que ofrece el protocolo USB 2.0 en high-speed es posible realizar un sistema soft real time de precisión de milisegundos con jitter de microsegundos (por ejemplo adecuado para el control de motor de pasos que realizan estimulación mecánica o lumínica) sin embargo no resulta suficiente para la ejecución de ciclos cerrados de la misma precisión (por ejemplo para protocolos de creación de canales iónicos o sinápticos mediante dynamic-clamp que requieren de muy alta precisión temporal). Si es posible implementar ciclos de esta precisión con tarjetas de mayor coste y menor portabilidad, como por ejemplo las DAQ usadas en el GNB de la serie M PCI de National Instruments.

6.2 Trabajo futuro

El protocolo USB es un modo de conexión de periféricos muy extendido y con gran potencial en la actualidad. Una gran variedad de sistemas han evolucionado hacia la conectividad mediante el protocolo USB (ratones de ordenador, videocámaras, micrófonos, unidades de almacenamiento, etc.) e incluso las grandes compañías del sector de la monitorización, medición y control como National Instrumnet ofrecen soluciones con tarjetas de adquisición de datos USB. La investigación in situ ofrece muchas posibilidades nuevas, pero tiene sus requisitos específicos y cada vez es más necesario dotar de dinamismo a los sistemas. La creación de un sistema USB que se ejecute en hard real time es necesario para muchas áreas de aplicación. En la actualidad existe un proyecto que ha logrado una pila USB para el host en un sistema embebido llamado OnTime RTOS, www.on-time.com/. RTUSB-32 - Real-Time Embedded USB Host Stack habilita la posibilidad de realizar transmisiones de datos cada frame o cada microframe de manera periódica o asimétrica. Este proyecto trabaja bajo el entorno Windows y su explotación requiere de licencia. En el software libre y en el entorno Linux no se ofrecen soluciones tan especializadas para este tipo de problemas. El controlador que nos ofrece Comedi para el manejo de las DAQ está diseñado para su uso en entornos de propósito general Linux. El protocolo USB sobre RTAI solo puede ser ejecutado en modo soft real time ya que no tiene garantías de hard real time. Existen proyectos como USB20RT, que intentan dar funcionalidad y garantías de latencias fijas para el manejo de las tramas USB por parte del host PC. El proyecto se encuentra en fase alfa y no ha sido testado todavía para el modo isócrono de transferencia. Si se consigue hacer funcionar el driver de la DAQ USBDUX sobre una pila USB que permita la ejecución de las tareas en modo hard real time, el sistema ofrecerá mejores características temporales y se podrán realizar un sistema en tiempo real mucho más fiable.

En el protocolo USB, en versión de super alta velocidad 3.0, las velocidades que se manejan son muy superiores a las de la versión 2.0. En esta versión 3.0 se obtienen hasta 600 MB/s, 10 veces más que las velocidades teóricas que alcanza la versión 2.0. En la versión 3.0 se utilizan las mismos modos de transmisión por lo que el sistema debe seguir considerándose soft real time en cualquiera de sus configuraciones, tanto en modo de transferencia isócrono (se pueden producir errores sin ser detectados) o en modo bulk o por interrupciones (no se asegura latencia fija de transmisión). Estas DAQ's están desarrolladas para trabajar en la versión 2.0 del protocolo USB, en futuras DAQ's en versión 3.0 se podrá realizar la exploración de drivers capaces de implementar sistemas con mayor precisión temporal. Con una mayor tasa de transferencia del protocolo USB, favoreciendo así que la comunicación DAQ/PC sea menos crítico y bajo las condiciones de tiempo real (PC host con garantías de control estricto temporal) se podrán crear sistemas que empleen protocolos de estimulación dependientes de actividad en la escala de milisegundos sin sufrir jitter.

En la configuración actual en un PC de varios procesadores con RTAI y las librerías Comedi, en las tarjetas USBDUX y el protocolo USB encontramos un mecanismo de precisión de milisegundos que permite la adquisición y la estimulación de sistemas biológicos que permitan realizar experimentación en el GNB. Un ejemplo del uso que se podrá dar al sistema es en la elaboración de un sistema de control de LEDs para la estimulación visual en interfaces cerebro-máquina, proyecto que se llevará a cabo en el GNB el próximo año.

Referencias

Axelson, J. (2005). *USB Complete: Everything You Need to Develop Custom USB Peripherals;*

Chamorro, P., Levi, R., Rodriguez, F. B., Pinto, R. D., and Varona, P. (2009). Real-time activity-dependent drug microinjection. *BMC Neurosci.* 10, P296.

Chamorro, P., Muñiz, C., Levi, R., Arroyo, D., Rodríguez, F. B., and Varona, P. (2012a). Generalization of the dynamic clamp concept in neurophysiology and behavior. *PLoS One* 7, e40887.

Chamorro, P., Muñiz, C., Levi, R., Arroyo, D., Rodríguez, F. B., and Varona, P. (2012b). Generalization of the dynamic clamp concept in neurophysiology and behavior. *PLoS One* 7, e40887.

Computing, M. Fundamental Signal Conditioning. 1–20.

Destexhe, A., and Bal, T. (2009). *Dynamic-clamp: from principles to applications*. , eds. A. Destexhe and T. Bal Springer, New York.

Dozio, L., and Mantegazza, P. (2003). Linux Real Time Application Interface (RTAI) in low cost high performance motion control. *Motion Control*.

Fernandez-Vargas, J., Pfaff, H. U., Rodriguez, F. B., and Varona, P. (2013). Assisted closed-loop optimization of SSVEP-BCI efficiency. *Front. Neural Circuits* 7, Article 27.

Hansson, H., Carlson, J., Isovic, D., Lundqvist, K., Nolte, T., and Ouimet, M. (2010). *Real-Time Systems*. , eds. P. Pettersson, S. Punnekkat, and C. Seceleanu MIT.

Korver, N. (2003). University of Twente Bus for real-time systems.

Mannori, S., Nikoukhah, R., and Steer, S. (2006). Free and open source software for industrial process control systems.

Mantegazza, P. (2006). DIAPM RTAI-Beginner's Guide. *RTAI Doc. Artic. http//www. rtai. org/24 th …*.

Muniz, C., Levi, R., Benkrid, M., Rodriguez, F. B., and Varona, P. (2008). Real-time control of stepper motors for mechano-sensory stimulation. *J. Neurosci. Methods* 172, 105–111.

Muniz, C., Rodriiguez, F. B., and Varona, P. (2009). RTBiomanager: a software platform to expand the applications of real-time technology in neuroscience. *BMC Neurosci.* 10, P49.

Muñiz, C. (2005). Advanced Dynamic Clamp: Novel techniques and protocols for realistic stimulation in neuronal systems.

Racciu, G., and Mantegazza, P. (2006). RTAI User Manual 3.4.

Ripoll Ripoll, J. I. (1997). Planificación en sistemas de Tiempo Real Aspectos básicos de la problemática de planificación para Tiempo−Real. Principales resultados de la teoría de prioridades estáticas.

Robinson, H., and Kawai, N. (1993). Injection of digitally synthesized synaptic conductance transients to measure the integrative properties of neurons. *J. Neurosci. Methods* 49, 157.

Sharp, A., O'Neil, M., Abbott, L., and Marder, E. (1993). The dynamic clamp: artificial conductances in biological neurons. *Trends Neurosci.* 16, 389.

Glosario

API	Application Programming Interface
DAQ	Data acquisition
USB	Universal Serial Bus
RTAI	Real Time Aplication Interface
Comedi	Control and measurement device interface
PC	Personal computer

A Manual de instalación

INSTALACIÓN RTAI

Documentación de los Scripts.

21 de Febrero 2012

En este manual está elaborado por el GNB para la instalación de RTAI y Comedi en un kernel genérico 2.6. El manual está compuesto por 9 scripts, en cada script se realiza una explicación de los pasos que se llevan a cabo. Se ha validado en la versión 10.04 de Ubuntu.

Script 1 (script1.sh):

El primer script se encarga de instalar todas las cosas necesarias para la compilación y posterior instalación de todo lo demás.

```
sudo apt-get update
echo Y|sudo apt-get install cvs subversion build-essential
echo Y|sudo apt-get install kernel-package linux-source libncurses5-dev
echo Y|sudo apt-get install libtool automake
echo Y|sudo apt-get install bison flex
echo Y|sudo apt-get install libboost-dev libboost-program-options-dev libgsl0-dev
echo Y|sudo apt-get install gfortran sablotron tcl8.5-dev tk8.5-dev xaw3dg-dev libpvm3 pvm-dev
libgtkhtml2-dev libzvt-dev libvte-dev
echo Y|sudo apt-get install qt3-apps-dev
echo Y|sudo apt-get install qt3-assistant
echo Y|sudo apt-get install qt3-designer
echo Y|sudo apt-get install qt3-dev-tools
echo Y|sudo apt-get install qt3-tools
echo Y|sudo apt-get install libqt4-dev libqwt5-qt4-dev
echo Y|sudo apt-get install cmake
echo Y|sudo apt-get install fakeroot
echo Y|sudo apt-get install git-core
sudo reboot
```

Script 2 (script2.sh)

```
cd /usr/src
sudo wget http://www.kernel.org/pub/linux/kernel/v2.6/linux-2.6.35.7tar.bz2
sudo tar xjvf linux-2.6.35.7tar.bz2
sudo ln -s linux-2.6.35.7 linux
echo "Extraccion de kernel completa "
sleep 5
```

En esta primera parte nos descargamos el kernel sobre el que vamos a instalar rtai. Hay que asegurarse, que el script tiene escrita la versión que realmente queremos instalar y no una anterior (o posterior). Una vez descargado, lo descomprimimos y creamos un enlace a la carpeta llamado linux. Esto será util más adelante, para no tener que estar cambiando el nombre del directorio en el resto de los scripts.

***Este script en concreto descarga la versión 2.6.35.7 desde kernel.org.*

```
cd /opt
sudo cvs -d:pserver:anonymous@cvs.gna.org:/cvs/rtai co magma
sudo ln -s magma rtai
echo "Descarga de rtai completa "
sleep 5
cd /opt
sudo wget --no-check-certificate https://www.rtai.org/RTAI/rtai-3.9-test2.tar.bz2
sudo tar xjvf rtai-3.9-test2.tar.bz2
sudo ln -s rtai-3.9-test2 rtai
echo "Extraccion de RTAI completa "
sleep 5
```

Esta parte descarga rtai, lo descomprime y crea un enlace llamado rtai. Al igual que con la versión del kernel, debemos asegurarnos de que nos bajamos la versión adecuada de rtai, y de que esta es compatible con el kernel que nos hemos descargado.

**Este script en concreto, descarga la version 3.9-test2 de rtai. Que es compatible con la versión de kernel 2.6.35.7 que es la que habíamos descargado antes.

```
cd /opt
git clone git://comedi.org/git/comedi/comedi.git

git clone git://comedi.org/git/comedi/comedilib.git
git clone git://comedi.org/git/comedi/comedi_calibrate.git
git clone git://comedi.org/git/comedi/comedi-nonfree-firmware.git
echo "Descarga de librerias comedilib completa "
sleep 5
```

Descargamos las librerías comedi usando git. Hay que comprobar que se han descargado correctamente. En caso contrario, descargarlas de forma manual desde: http://www.comedi.org/download.html. Descomprimidlas y crear los links (con el comando ln -s, exactamente igual que se hace para rtai y el kernel de linux) a las carpetas comedi, comedilib, comedi_calibrate y comedi-nonfree-firmware

```
cd /opt
sudo wget http://www.scilab.org/download/4.1.2/scilab-4.1.2-src.tar.gz
sudo wget http://www.scilab.org/download/4.1.2/man-eng-scilab-4.1.2.zip
echo "Descarga de scilab y manual completa "
sleep 5
cd /opt
sudo svn co https://qrtailab.svn.sourceforge.net/svnroot/qrtailab/trunk qrtailab
echo "Descarga de librerias de qrtailab completa "
sleep 5
cd /opt
sudo wget http://downloads.sourceforge.net/qrtailab/QRtaiLab-0.1.6.tar.gz
sudo tar xvzf QRtaiLab-0.1.6.tar.gz
echo "Extraccion de QrtaiLab completa "
sleep 5
```

Descargamos scilab, QrtaiLab y las librerías qrtailab, para su posterior instalación.

```
cd /usr/src/linux
echo "Vamos a parchear 1"
sudo su
```

Nos situamos en la carpeta de linux y nos logueamos como root, para parchear el kernel.

Script 3 (script3.sh)

```
cd /usr/src/linux
echo "Vamos a parchear 2"
sleep 5
patch -p1 < /opt/rtai/base/arch/x86/patches/hal-linux-2.6.35.7-x86-2.8-01.patch
echo "Patch de kernel completo "
sleep 5
```

Nos volvemos a meter en la carpeta de linux y parcheamos el Kernel descargado. A la hora de parchear, hay que asegurarse de utilizar el parche adecuado para el kernel que estamos intentando instalar. Hay por lo menos un parche para cada versión de kernel compatible con rtai. Y(Si has seguido los pasos explicados hasta ahora) podrás encontrarlos todos en el directorio /opt/rtai/base/arch/x86/patches/ busca el que sea más adecuado para tu kernel y cámbialo en el script.

***En este caso, usamos el parche "hal-linux-2.6.35.7-x86-2.8-01.patch " que es adecuado para la version 2.6.35.7 del kernel.*

```
cd /usr/src/linux
cp /boot/config-2.6.32-38-generic .config
sudo cp .config .config.old
```

Copiamos el archivo de configuración de nuestra versión actual del kernel (Luego ya lo modificaremos) en linux. Y además, hacemos una copia en .config.old para poder modificar el archivo de configuración sin perder lo que había antes, por si necesitamos volver a las opciones originales.

```
cd /usr/src/linux
sudo make clean
sudo make mrproper
sudo make menuconfig
echo "Completado menuconfig numero 1 "
sleep 5
```

En el menú de configuración hay que asegurarse de las siguientes cosas:

o Processor type and features -> Symmetric multi-processing support (multiprocessor)
o Processor type and features-> Processor family (Que sea adecuada, probablemente la última intel core que haya)
o Enable loadable module support > Module versioning support = no
o Processor type and features > HPET = no
o Processor type and features > Interrupt pipeline = yes
o Power management options > Power Management support = no
o Power management options > CPU Frequency scaling > CPU Frequency scaling = no"
o Device Drivers -> Network device support -> Ethernet (XXXX Mbit) = yes
o Device Drivers -> Network device support -> Wlan =yes
o Device Drivers -> Serial ATA and Parallel ATA driver -> AHCI SATA support (Asegurarse de que está seleccionado y no como módulo)
o Device Drivers -> Graphics support -> /dev/agpgart (AGP Support) =yes
o Device Drivers -> Sound card support -> Advanced Linux Sound Architecture -> PCI sound devices=yes

Desactivar todas las opciones marcadas en Kernel Hacking. (Ojo, desmarcarlas dos veces, ya que al desmarcar una subopcion, te permite desmarcar opciones que antes no podías).

Script 4 (script4.sh)

```
cd /usr/src/linux
sudo make-kpkg clean
sudo CONCURRENCY_LEVEL=4 fakeroot make-kpkg -initrd -append-to-version=-rtai kernel_image
kernel_headers
echo "Compilacion de kernel completo "
sleep 5
```

Ahora vamos con la instalación del Kernel. El comando make-kpkg compila el kerel y te lo agrupa en un paquete, facil de instalar y desinstalar. Fakeroot, por su parte, es una librería que hace que el comando tar crea que está siendo invocado como root.

Si tienes más de un procesador, CONCURRENCY_LEVEL=*#procesadores*, acelera las cosas compilando en paralelo.

El flag **append-to-version=-rtai** indica que al nombre de la imagen, ha de añadírsele al final la versión de rtai, además de la versión de kernel. Esto es para que a la hora de arrancar, sepas que kernels tienes con rtai y cuales no.

Las opciones, **kernel_image** y **kernel_headers** indican que ademas se van a crear los paquetes para la instalación de la imagen y las cabeceras del kernel.

```
cd /usr/src
sudo dpkg -i linux-headers-2.6.35.7-rtai_2.6.35.7-rtai-10.00.Custom_i386.deb
sudo dpkg -i linux-image-2.6.35.7-rtai_2.6.35.7-rtai-10.00.Custom_i386.deb
echo "Instalacion del kernel completado "
sleep 5
sudo reboot
```

Insatalamos los paquetes de imagen y cabeceras que habíamos creado antes. (Hay que asegurarse que el archivo que nos ha creado la anterior compilación y empaquetación es el que hay en el script, si no, hay que cambiar los hombres. Los nombres de los paquetes serán del tipo:

linux-headers-*version de linux-procesador*.deb (Para las cabeceras)
linux-image-*version de linux-procesador*.deb (Para la imagen)

La instalación de la imagen te actualiza también el grub. Reiniciamos e iniciamos sesión con el kernel parcheado.

Script 5 (script5.sh)

```
cd /opt/rtai
sudo make menuconfig
echo "Menu de configuracion segundo completado "
sleep 5
```

Hacemos el menú de configuración de rtai. En el menú de configuración hay que asegurarse de las siguientes opciones:

- *Installation: /usr/realtime*
- *Kernel source: /usr/src/linux*
- *En Machine, elge el número CPUs (compruebalo introduciendo cat /proc/cpuinfo por línea de comandos y mirando cuantos procesadores aparecen listados)*
- *Nota: con el Kernel 2.26.3 era necesario desseleccionar "In-Kernel C++ Support" en "Add-Ons"*

```
sudo make
sudo make install
sudo sed -i 's/(PATH=")/1/usr/realtime/bin:/' /etc/environment
export PATH=/usr/realtime/bin:$PATH
echo "Nuevo PATH añadido "
sleep 5
```

Compilamos e instalamos rtai y añadimos el PATH de rtai.

```
cd /opt/comedi
sudo sh autogen.sh
sudo ./configure --with-linuxdir=/usr/src/linux --with-rtaidir=/usr/realtime
sudo make
sudo make install
sudo make dev
echo "Instalacion de RTAI completa "
sleep 5
```

Compilamos e instalamos las comedi.

```
sudo su
```

Nos logueamos como superusuarios para los siguientes pasos.

Script 6 (script6.sh)

```
echo 'options comedi comedi_num_legacy_minors=4' > /etc/modprobe.d/comedi
##############################
cd /opt/comedilib
sudo sh autogen.sh
sudo ./configure
sudo make
sudo make install
sudo mkdir /usr/local/include/linux
echo "Comedilib completo "
sleep 5
##############################
cd /opt/comedi_calibrate
sudo autoreconf -i -B m4
sudo ./configure
sudo make
sudo make install
echo "Comedi calibrate completo "
sleep 5
```

Configuramos, compilamos e instalamos las librerías comedi y el comedi calibrate.

```
sudo cp /opt/comedi/include/linux/comedi.h /usr/local/include/
sudo cp /opt/comedi/include/linux/comedilib.h /usr/local/include/
sudo ln -s /usr/local/include/comedi.h /usr/local/include/linux/comedi.h
sudo ln -s /usr/local/include/comedilib.h /usr/local/include/linux/comedilib.h
cd /opt/rtai
sudo make menuconfig
echo "Menu de configuracion tercero completado "
sleep 5
```

Copiamos las cabeceras a la carpeta de includes y creamos los enlaces a las librerías. Rehacemos el menú de configuración de rtai, para añadir la compatibilidad con las comedi.En el menú de configuración, nos aseguramos, de que en *"Add-Ons",* esté marcada la opción *"Real Time COMEDI support in user space"*

Script 7 (script7.sh)

```
sudo make
sudo make install
```

Recompilamos y reinstalamos rtai con el nuevo menú de configuración, para poder hacer uso de las cómedi bajo rtai.

```
cd /opt
sudo tar xvzf scilab-4.1.2-src.tar.gz
sudo unzip man-eng-scilab-4.1.2.zip
sudo mv man-eng-scilab-4.1.2 scilab-4.1.2/man/eng/
cd scilab-4.1.2
sudo ./configure --with-gfortran --with-tk --with-gtk2
sudo make all
sudo ln -s /opt/scilab-4.1.2/bin/scilab /usr/local/bin/scilab
##############################
cd /opt/qrtailab
sudo sed 's/LIBS += -lqwt/LIBS += -lqwt-qt4/' qrtailab.config
sudo sed 's/INCLUDEPATH += . /usr/realtime/include /usr/include/qwt//INCLUDEPATH += .
/usr/realtime/include /usr/include/qwt-qt4/' qrtailab.config
sudo qmake-qt41
sudo make
sudo make install
```

Antes de compilar, modificamos el fichero de configuración par aque qrtailab funcione con las qt4. Compilamos, configuramos e Instalamos scilab y qrtailab.

Script 8 (script8.sh)

Configuramos las cabeceras de las librerías comedi e instalamos la librería matemática.

```
cd /usr/local/include
sudo mv comedi.h comedi.h.old
sudo mv comedilib.h comedilib.h.old
sudo cp /opt/comedilib/include/comedilib.h .
sudo cp /opt/comedilib/include/comedi.h .
##########LIBRERIA MATEMATICA#######################
cd /opt/rtai/base/math
sudo make
sudo make install
#########COMEDI LIBRARIES#########################
cd /usr/local
sudo mkdir comedi
cd  comedi
sudo mkdir include
cd include
sudo mkdir linux
cd linux
sudo ln -sf /opt/comedi/include/linux/comedi.h
sudo ln -sf /opt/comedi/include/linux/comedilib.h
sudo ldconfig
```

Script 9 (script9.sh)

```
#Instalación de Open CV
sudo apt-get install libgtk2.0-dev
cd /opt
sudo mkdir opencv.build
cd opencv.build
sudo wget http://sourceforge.net/projects/opencvlibrary/files/opencv-unix/2.1/OpenCV-2.1.0.tar.bz2
sudo tar -xjvf  OpenCV-2.1.0.tar.bz2
sudo mkdir release
cd release
sudo cmake -D CMAKE_BUILD_TYPE=RELEASE -D CMAKE_INSTALL_PREFIX=/usr/local -D
BUILD_PYTHON_SUPPORT=ON ..
sudo make
sudo make install
export LD_LIBRARY_PATH=/usr/local/lib:$LD_LIBRARY_PATH
```

Instalamos la librería libgtk2.0-dev necesaria para trabajar con OpenCV, nos descargamos OpenCV (En este script en la versión 2.1), lo configuramos, lo compilamos y lo instalamos. Para probar que se ha instalado bien puedes ir a la carpeta *cd /opt/opencv.build/trunk/opencv/release/bin* y correr el programa de prueba ./cxcoretest

```
#Instalación de qwt
cd /usr/local
sudo wget http://sourceforge.net/projects/qwt/files/qwt/5.2.2/qwt-5.2.2.zip
sudo unzip qwt-5.2.2.zip
cd qwt-5.2.2
sudo qmake
sudo make
sudo make install
sudo echo export LD_LIBRARY_PATH=/usr/local/qwt-5.2.2/lib/ >>  ~/.bashrc
```

Instalamos la librería qwt y modificamos el archivo ~/.bashrc para que haga el export de la ruta automáticamente al arrancar el ordenador.

B Script de inicialización del sistema RTAI y de las USBDUX.

```
#!/bin/bash
if test \! -c $DESTDIR/dev/rtai_shm; then \
    echo "Creating rtai shared memory"
    mknod -m 666 $DESTDIR/dev/rtai_shm c 10 254; \
fi;


for n in `seq 0 9`; do \
  if test \! -c $DESTDIR/dev/rtf$n; then \
    echo "Creating real time fifo $n"
    mknod -m 666  $DESTDIR/dev/rtf$n c 150 $n; \
  fi; \
done;

for i in `seq 0 1`; do \
    if test \! -c $DESTDIR/dev/comedi$i; then \
      echo "Creating comedi device $i"
      mknod -m 666 /dev/comedi$i c 98 $i; \
    fi; \
done;

echo "Creating devices ........ OK"

export PATH=/usr/realtime/bin:$PATH
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/usr/realtime/lib

prefix=`rtai-config --prefix`

if test "$prefix" = ""; then
echo "ERROR: please set your PATH variable to <rtai-install>/bin"
exit
fi

${DESTDIR}/usr/realtime/bin/rtai-load
rmmod comedi
insmod /lib/modules/2.6.35.7-rtai/comedi/comedi.ko
insmod /lib/modules/2.6.35.7-rtai/comedi/kcomedilib/kcomedilib.ko
insmod /lib/modules/2.6.35.7-rtai/comedi/drivers/comedi_fc.ko
insmod /usr/realtime/modules/rtai_comedi.ko
insmod /lib/modules/2.6.35.7-rtai/comedi/drivers/usbduxsigma.ko
insmod /lib/modules/2.6.35.7-rtai/comedi/drivers/usbduxfast.ko

/home/dynamic/Desktop/usbduxsigma_driver-
lts/comedi/drivers/usbduxsigma_firmware.bin -v /dev/comedi0
usbduxsigma
/opt/comedilib/etc/hotplug/usb/usbduxfast/usbduxfast_firmware.bin
-v /dev/comedi0 usbduxfast
```

C Plantilla GNUPLOT

```
#!/usr/bin/gnuplot -persist

set terminal postscript color enhanced
set output "RTAIusbduxsigmaadquisicion1khz.eps"

set terminal  postscript  color enhanced
set output "periodoiroshima.eps"

set termoption enhanced
set encoding iso_8859_1

set xdata
set ydata

set title "  Grabaci{\363}n de la actividad de la neurona
artificial"
set title  offset character 0, 0, 0 font "" norotate
set xlabel " # Muestra"
#set xlabel " # Ciclo"
set xlabel  offset character 0, 0, 0 font "" textcolor lt -1
norotate

set ylabel "Tensi{\363}n(v)"
#set ylabel "Tiempo(s)"
set ylabel  offset character 0, 0, 0 font "" textcolor lt -1
rotate by 90

plot [][]  "fichero.dat"  using 1 w l title 'adquisici{\363}n'
pause -1
#    EOF
```

D Manual del programador

Manual del programador

En este apartado se aporta la información relevante de las DAQ's USBDUX, sus conexiones en el conector HD-42 pines, el esquemático detallado ofrecido por el fabricante para cada DAQ, la salida que genera la función comedi_test y el código C utilizado en cada DAQ para la realización de las tareas previstas para cada tarjeta. En el caso de la USBDUX-fast como se emplea únicamente para la grabación del funcionamiento del sistema en conjunto solo emplea la adquisición asíncrona sobre Comedilib. En la USBDUX-sigma se realizan pruebas de monitorización, estimulación y por último ciclos cerrados gobernados por objetivo en entrono de tiempo real y de propósito general n configuración síncrona o asíncrona.

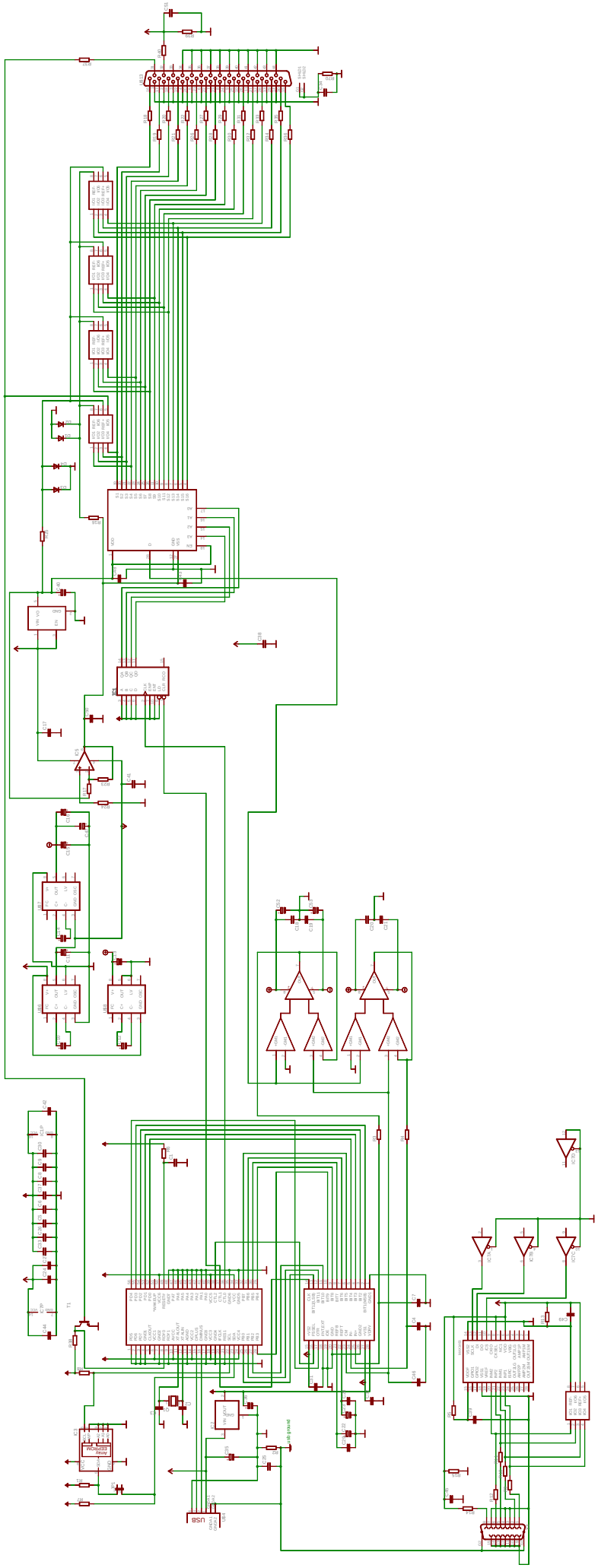
1. USBDUX-fast

1.1. Pinouts

44 pin HD connector (f)

Pin	Direction	Function
1-14	GND	
15	I	A/D channel 15
16	I	A/D channel 0
17	I	A/D channel 1
18	I	A/D channel 2
19	I	A/D channel 3
20	I	A/D channel 4
21	I	A/D channel 5
22	I	A/D channel 6
23	I	A/D channel 7
24	I	A/D channel 8
25	I	A/D channel 9
26	I	A/D channel 10
27	I	A/D channel 11
28	I	A/D channel 12
29	I	A/D channel 13
30	I	A/D channel 14
31	I	external trigger
32	Supply	$+5V$
33	NC	not conn
34-44	GND	

1.2. Esquemático de la DAQ USBDUX-fast



1.3. Comedi_test

Salida de la función comedi_test realizada sobre la tarjeta DAQ USBDUX-fast, esta función nos revela las características del device y subdevices.

```
I: Comedi version: 0.7.76
I: Comedilib version: unknown =)
I: driver name: usbduxfast
I: device name: usbduxfast
I:
I: subdevice 0
I: testing info...
rev 1
I: subdevice type: 1 (analog input)
  number of channels: 16
  max data value: 4096
  ranges:
    all chans: [-0.75,0.75] [-0.5,0.5]
I: testing insn_read...
rev 1
comedi_do_insn returned 1, good
I: testing insn_read_0...
comedi_do_insn returned 0, good
I: testing insn_read_time...
rev 1
comedi_do_insn: 3
read time: 1677 us
I: testing cmd_no_cmd...
not applicable
I: testing cmd_probe_src_mask...
rev 1
command source mask:
  start: now|ext|int
  scan_begin: follow|timer|ext
  convert: timer|ext
  scan_end: count
  stop: none|count
I: testing cmd_probe_fast_1chan...
command fast 1chan:
  start: now 0
  scan_begin: follow 0
  convert: timer 33
  scan_end: count 1
  stop: count 2
I: testing cmd_read_fast_1chan...
I: testing cmd_write_fast_1chan...
not applicable
I: testing cmd_logic_bug...
rev 1
command_test returned 1, good
I: testing cmd_fifo_depth_check...
```

64, 1
128, 1
256, 1
512, 2
1024, 4
2048, 8
4096, 16
8192, 32
16384, 64
32768, 128
I: testing cmd_start_inttrig...
I: testing mmap...
0xb7700000 ok
0xb7701000 ok
0xb7702000 ok
0xb7703000 ok
0xb7704000 ok
compare ok
0xb7700000 segfaulted (ok)
0xb7701000 segfaulted (ok)
0xb7702000 segfaulted (ok)
0xb7703000 segfaulted (ok)
0xb7704000 segfaulted (ok)
I: testing read_select...
I: testing bufconfig...
buffer size 2097152
max buffer size 20971520
setting buffer size to 4096
buffer size set to 4096
buffer size now at 4096
setting buffer size past limit, 20975616
got EPERM, good
setting buffer size to max, 20971520
buffer size now at 20971520

1.4. Programa de adquisición asíncrona en modo usuario con comedilib.

```
/*
 * Example of using commands - asynchronous input
 * Part of Comedilib
 *
 * Copyright (c) 1999,2000,2001 David A. Schleef <ds@schleef.org>
 *               2008 Bernd Porr <berndporr@f2s.com>
 *
 * This file may be freely modified, distributed, and combined with
 * other software, as long as proper attribution is given in the
 * source code.
 */

/*
 * The program is used to test the usbdux sigma board
 */

/*  El  mismo  programa  vale  para  usbdux  fast,  se  configuran  sus
paramentos*\

#include <stdio.h>
#include <rtai_comedi.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/time.h>
#include <errno.h>
#include <stdlib.h>
#include <string.h>

#include <rtai_lxrt.h>
extern comedi_t *device;

struct parsed_options
{
      char *filename;
      double value;
      int subdevice;
      int channel;
      int aref;
      int range;
      int physical;
      int verbose;
      int n_chan;
      int n_scan;
      double freq;
};



#define BUFSZ 10000
char buf[BUFSZ];

#define N_CHANS 256
static unsigned int chanlist[N_CHANS];
static comedi_range * range_info[N_CHANS];
static lsampl_t maxdata[N_CHANS];
```

```
int prepare_cmd(comedi_t *dev, int subdevice, int n_scan, int n_chan,
unsigned period_nanosec, comedi_cmd *cmd);

int prepare_cmd_lib(comedi_t *dev, int subdevice, int n_scan, int n_chan,
unsigned period_nanosec, comedi_cmd *cmd);

void do_cmd(comedi_t *dev,comedi_cmd *cmd);

void print_datum(lsampl_t raw, int channel_index, short physical);

char *cmdtest_messages[]={
      "success",
      "invalid source",
      "source conflict",
      "invalid argument",
      "argument conflict",
      "invalid chanlist",
};

int main(int argc, char *argv[])
{
      comedi_t *dev;
      comedi_cmd c,*cmd=&c;
      int ret;
      int total=0;
      int i;
      struct timeval start,end;
      int subdev_flags;
      lsampl_t raw;

      struct parsed_options options;

      /* The following variables used in this demo
       * can be modified by command line
       * options.  When modifying this demo, you may want to
       * change them here. */
      options.filename = "/dev/comedi0";
      options.subdevice = 0;
      options.channel = 0;
      options.range = 0;
      options.aref = AREF_GROUND;
      options.n_chan = 1;
      options.n_scan = 1000;
      options.freq = 1000.0;

      /* open the device */
      dev = comedi_open(options.filename);
      if(!dev){
            comedi_perror(options.filename);
            exit(1);
      }

      // Print numbers for clipped inputs
      comedi_set_global_oor_behavior(COMEDI_OOR_NUMBER);

      /* Set up channel list */
      for(i = 0; i < options.n_chan; i++){
            chanlist[i]  =  CR_PACK(options.channel  +  i,  options.range,
options.aref);
            range_info[i]   =   comedi_get_range(dev,   options.subdevice,
options.channel, options.range);
```

```
            maxdata[i]   =   comedi_get_maxdata(dev,   options.subdevice,
options.channel);
      }

      /* prepare_cmd_lib() uses a Comedilib routine to find a
       * good command for the device.  prepare_cmd() explicitly
       * creates a command, which may not work for your device. */
      //prepare_cmd_lib(dev,      options.subdevice,      options.n_scan,
options.n_chan, 1e9 / options.freq, cmd);
      prepare_cmd(dev, options.subdevice, options.n_scan, options.n_chan,
1e9 / options.freq, cmd);

      /* comedi_command_test() tests a command to see if the
       * trigger sources and arguments are valid for the subdevice.
       * If a trigger source is invalid, it will be logically ANDed
       * with valid values (trigger sources are actually bitmasks),
       * which may or may not result in a valid trigger source.
       * If an argument is invalid, it will be adjusted to the
       * nearest valid value.  In this way, for many commands, you
       * can test it multiple times until it passes.  Typically,
       * if you can't get a valid command in two tests, the original
       * command wasn't specified very well. */
      ret = comedi_command_test(dev, cmd);
      if(ret < 0){
            comedi_perror("comedi_command_test");
            if(errno == EIO){
                  fprintf(stderr,"Ummm... this subdevice doesn't support
commands\n");
            }
            exit(1);
      }
      ret = comedi_command_test(dev, cmd);
      if(ret < 0){
            comedi_perror("comedi_command_test");
            exit(1);
      }
      fprintf(stderr,"second test returned %d (%s)\n", ret,
                  cmdtest_messages[ret]);
      if(ret!=0){
            fprintf(stderr, "Error preparing command\n");
            exit(1);
      }

      /* this is only for informational purposes */
      gettimeofday(&start, NULL);
      fprintf(stderr,"start     time:     %ld.%06ld\n",     start.tv_sec,
start.tv_usec);

      /* start the command */
      ret = comedi_command(dev, cmd);
      if(ret < 0){
            comedi_perror("comedi_command");
            exit(1);
      }
      subdev_flags = comedi_get_subdevice_flags(dev, options.subdevice);


//while(1){}
      while(1){
            ret = read(comedi_fileno(dev),buf,BUFSZ);
            if(ret < 0){
```

```
                perror("read");
                break;
          }else if(ret == 0){

                break;
          }else{
                static int col = 0;
                int bytes_per_sample;
                total += ret;
                if(options.verbose)fprintf(stderr, "read %d %d\n", ret,
total);
                if(subdev_flags & SDF_LSAMPL)
                      bytes_per_sample = sizeof(lsampl_t);
                else
                      bytes_per_sample = sizeof(sampl_t);
                for(i = 0; i < ret / bytes_per_sample; i++){
                      if(subdev_flags & SDF_LSAMPL) {
                            raw = ((lsampl_t *)buf)[i];
                      } else {
                            raw = ((sampl_t *)buf)[i];
                      }
                      print_datum(raw, col, options.physical);
                      col++;
                      if(col == options.n_chan){
                            printf("\n");
                            col=0;
                      }
                }
          }
    }

    /* this is only for informational purposes */
    gettimeofday(&end,NULL);
    fprintf(stderr,"end time: %ld.%06ld\n", end.tv_sec, end.tv_usec);

    end.tv_sec -= start.tv_sec;
    if(end.tv_usec < start.tv_usec){
          end.tv_sec--;
          end.tv_usec += 1000000;
    }
    end.tv_usec -= start.tv_usec;
    fprintf(stderr,"time: %ld.%06ld\n", end.tv_sec, end.tv_usec);

    return 0;
}

/*
 * This prepares a command in a pretty generic way.  We ask the
 * library to create a stock command that supports periodic
 * sampling of data, then modify the parts we want. */
int prepare_cmd_lib(comedi_t *dev, int subdevice, int n_scan, int n_chan,
unsigned scan_period_nanosec, comedi_cmd *cmd)
{
    int ret;

    memset(cmd,0,sizeof(*cmd));

    /* This comedilib function will get us a generic timed
     * command for a particular board.  If it returns -1,
     * that's bad. */
```

```
	ret = comedi_get_cmd_generic_timed(dev, subdevice, cmd, n_chan,
scan_period_nanosec);
	if(ret<0){
		printf("comedi_get_cmd_generic_timed failed\n");
		return ret;
	}

	/* Modify parts of the command */
	cmd->chanlist = chanlist;
	cmd->chanlist_len = n_chan;
	if(cmd->stop_src == TRIG_COUNT) cmd->stop_arg = n_scan;

	return 0;
}

/*
 * Set up a command by hand.  This will not work on some devices.
 * There is no single command that will work on all devices.
 */
int prepare_cmd(comedi_t *dev, int subdevice, int n_scan, int n_chan,
unsigned period_nanosec, comedi_cmd *cmd)
{
	memset(cmd,0,sizeof(*cmd));

	/* the subdevice that the command is sent to */
	cmd->subdev =	subdevice;

	/* flags */
	cmd->flags = 0;

	/* Wake up at the end of every scan */
	//cmd->flags |= TRIG_WAKE_EOS;

	/* Use a real-time interrupt, if available */
	//cmd->flags |= TRIG_RT;

	/* each event requires a trigger, which is specified
	   by a source and an argument.  For example, to specify
	   an external digital line 3 as a source, you would use
	   src=TRIG_EXT and arg=3. */

	/* The start of acquisition is controlled by start_src.
	 * TRIG_NOW:	The start_src event occurs start_arg nanoseconds
	 *		after comedi_command() is called.  Currently,
	 *		only start_arg=0 is supported.
	 *  TRIG_FOLLOW:   (For an output device.)   The start_src event
occurs
	 *		when data is written to the buffer.
	 * TRIG_EXT:	start event occurs when an external trigger
	 *		signal occurs, e.g., a rising edge of a digital
	 *		line.  start_arg chooses the particular digital
	 *		line.
	 * TRIG_INT:	start event occurs on a Comedi internal signal,
	 *		which is typically caused by an INSN_TRIG
	 *		instruction.
	 */
	cmd->start_src =TRIG_NOW;
	cmd->start_arg =0;

	/* The timing of the beginning of each scan is controlled by
	 * scan_begin.
```

```
       * TRIG_TIMER:   scan_begin events occur periodically.
       *               The time between scan_begin events is
       *               convert_arg nanoseconds.
       * TRIG_EXT:     scan_begin events occur when an external trigger
       *               signal occurs, e.g., a rising edge of a digital
       *                    line.   scan_begin_arg chooses the particular
digital
       *               line.
       *  TRIG_FOLLOW:    scan_begin  events  occur  immediately  after  a
scan_end
       *               event occurs.
       * The scan_begin_arg that we use here may not be supported exactly
       * by the device, but it will be adjusted to the nearest supported
       * value by comedi_command_test(). */
      cmd->scan_begin_src =   TRIG_TIMER;
      cmd->scan_begin_arg = period_nanosec;          /* in ns */

      /*  The  timing  between  each  sample  in  a  scan  is  controlled  by
convert.
       * TRIG_TIMER:   Conversion events occur periodically.
       *               The time between convert events is
       *               convert_arg nanoseconds.
       * TRIG_EXT:     Conversion events occur when an external trigger
       *               signal occurs, e.g., a rising edge of a digital
       *               line.  convert_arg chooses the particular digital
       *               line.
       *  TRIG_NOW:           All  conversion  events  in  a  scan  occur
simultaneously.
       * Even though it is invalid, we specify 1 ns here.  It will be
       * adjusted later to a valid value by comedi_command_test() */
      cmd->convert_src =      TRIG_NOW;
      cmd->convert_arg =      0;          /* in ns */

      /* The end of each scan is almost always specified using
       * TRIG_COUNT, with the argument being the same as the
       * number of channels in the chanlist.  You could probably
       * find a device that allows something else, but it would
       * be strange. */
      cmd->scan_end_src =     TRIG_COUNT;
      cmd->scan_end_arg =     n_chan;           /* number of channels */

      /* The end of acquisition is controlled by stop_src and
       * stop_arg.
       * TRIG_COUNT:  stop acquisition after stop_arg scans.
       * TRIG_NONE:   continuous acquisition, until stopped using
       *              comedi_cancel()
       * */
      cmd->stop_src =         TRIG_COUNT;
      cmd->stop_arg =         n_scan;

      /* the channel list determined which channels are sampled.
         In general, chanlist_len is the same as scan_end_arg.  Most
         boards require this.  */
      cmd->chanlist =         chanlist;
      cmd->chanlist_len =     n_chan;

      return 0;
}

void print_datum(lsampl_t raw, int channel_index, short physical) {
      double physical_value;
```

```
	if(!physical) {
		printf("%d ",raw);
	} else {
		physical_value		=		comedi_to_phys(raw,
range_info[channel_index], maxdata[channel_index]);
		printf("%#8.6g ",physical_value);
	}
}
```

2. USBDUX-sigma

2.1. Información sobre la DAQ USBDUX-sigma

USBDUXsigma – www.linux-usb-daq.co.uk

1 Technical specs

1.1 A/D converter

- 16 channels
- 24 bit resolution
- Single ended
- Input range:
  - Bipolar: $-1.325 \dots 1.325V$ ($-1.25 \dots 1.25$)
- Input resistance: 10M
- Asynchronous sampling:
  - All channels simultaneously at a rate of 1kHz, 8 channels at 2kHz and 1-2 channels at 4kHz.
  - Latency is one sampling interval.
- Latency for synchronous sampling approx 2ms.

1.2 D/A converter

- 4 channels
- 8 bit resolution
- Output range:
  - Unipolar: $0 \dots 2.5V$
- Max output current: 1mA
- Latency approx 300us.

1.3 24 bit digital I/O

- Direction is programmable via comedi.
- PWM (8 channels or 4 channels with polarity bit)
- Latency for writes approx 300us and approx 500us for read operations.

1.4 Connectivity: USBDUX-D

- D connectors only
- Female 44 pin high density D connector electrically isolated for analogue I/O with +/-5V supply.
- Female 15 pin D connector for the digital I/O.

1.5 Physical Dimensions

- Enclosure: 144x90x30mm
- Board: 136x80x15mm

Latencies vary depending on system and USB bus load.

2 USB-DUX-SIGMA Pinouts

2.1 Analogue I/O

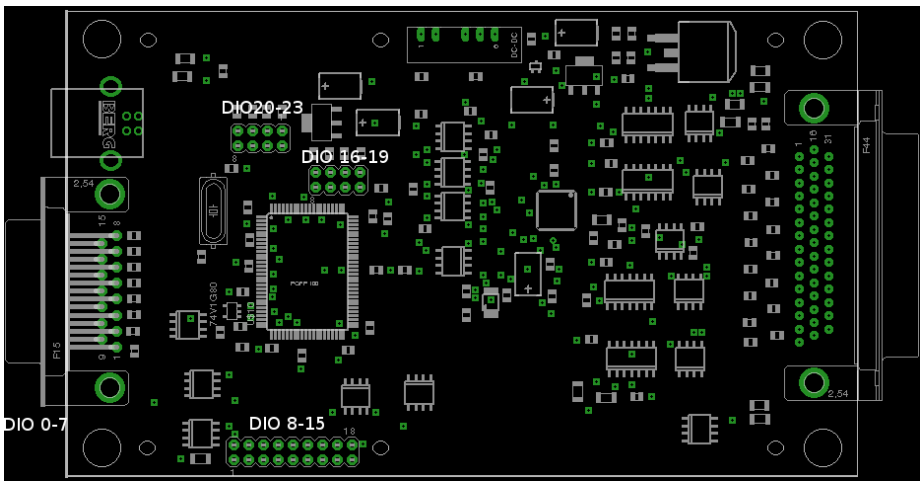
The analogue I/O have a 44 pin D connector (f). All signals and the supply output are electrically isolated from the USB and the digital I/O.

Pin	Direction	Function
1–14	GND	GND
15	I	A/D channel 15
16	I	A/D channel 0
17	I	A/D channel 1
18	I	A/D channel 2
19	I	A/D channel 3
20	I	A/D channel 4
21	I	A/D channel 5
22	I	A/D channel 6
23	I	A/D channel 7
24	I	A/D channel 8
25	I	A/D channel 9
26	I	A/D channel 10
27	I	A/D channel 11
28	I	A/D channel 12
29	I	A/D channel 13
30	I	A/D channel 14
31	O (!)	$-5V$
32	O (!)	$+5V$
33	N/C	N/C
34-40	GND	
41	O	D/A channel 0, unipol
42	O	D/A channel 1, unipol
43	O	D/A channel 2, unipol
44	O	D/A channel 3, unipol

Pins 31 and 32 are outputs (!) which can provide about 20mA of supply current for external circuitry such as amplifiers. It is advised to add 10μF decoupling capacitors because the voltage is unregulated and the internal DC/DC converter generates high frequency noise. The supply is provided through 51Ω resistors so that there will be a voltage drop when the current increases. Rail to rail amplifiers are recommended.

2.2 Digital I/O

There are a total of 24 digital I/O ports which are located at different positions on the board:



DIO 0-7 is available at the 15 pin D connector and the other DIO signals are available internally.

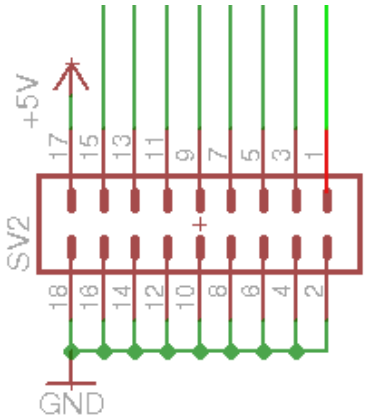
2.2.1 Bits 0-7

The I/O lines are 3.3V logic level and are protected by an diode array and 51Ω resistors.

Pin	Direction	Function
1	I/O	digital I/O bit 7
2	I/O	digital I/O bit 6
3	I/O	digital I/O bit 5
4	I/O	digital I/O bit 4
5	I/O	digital I/O bit 3
6	I/O	digital I/O bit 2
7	I/O	digital I/O bit 1
8	I/O	digital I/O bit 0
9–14	GND	
15	O (!)	NC/$5V$

2.2.2 Bits 8-15

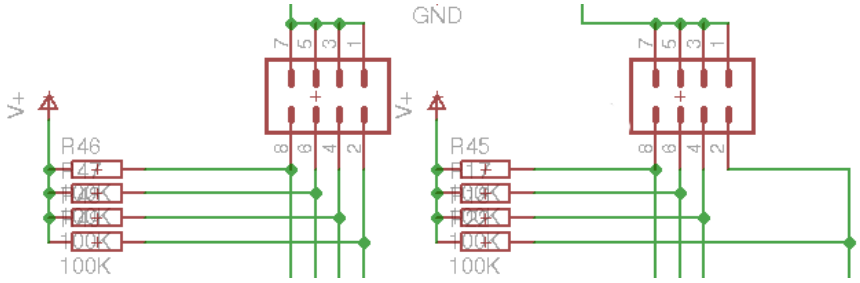
The inputs are available at a pin header internally. They are protected by an diode array. The logic is 3.3V.



Pin	Direction	Function
1	I/O	digital I/O bit 0
3	I/O	digital I/O bit 1
5	I/O	digital I/O bit 2
7	I/O	digital I/O bit 3
9	I/O	digital I/O bit 4
11	I/O	digital I/O bit 5
13	I/O	digital I/O bit 6
15	I/O	digital I/O bit 7
17	O (!)	NC/$5V$
2,4,6,. . . ,18	GND	

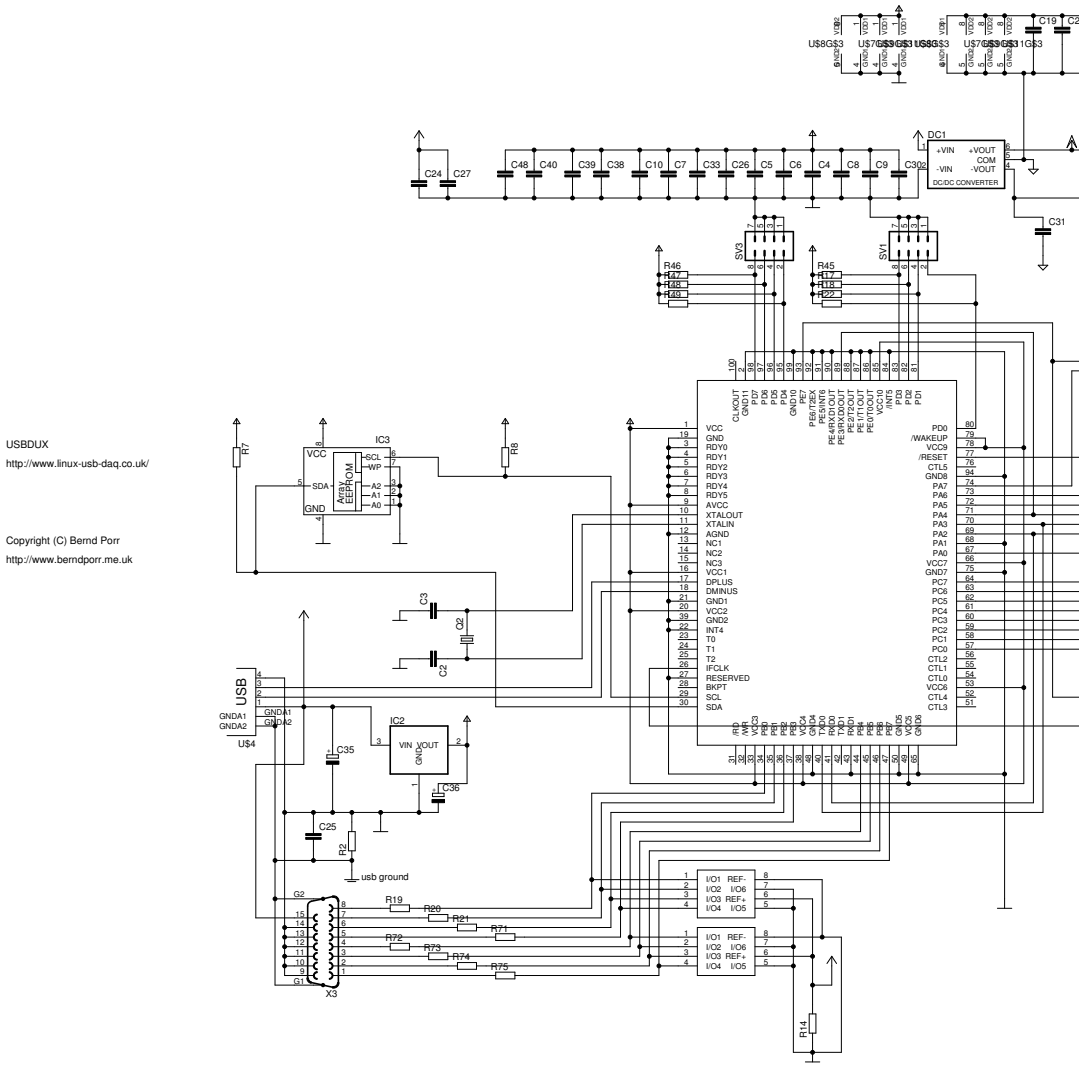
2.2.3 Bits 16-24

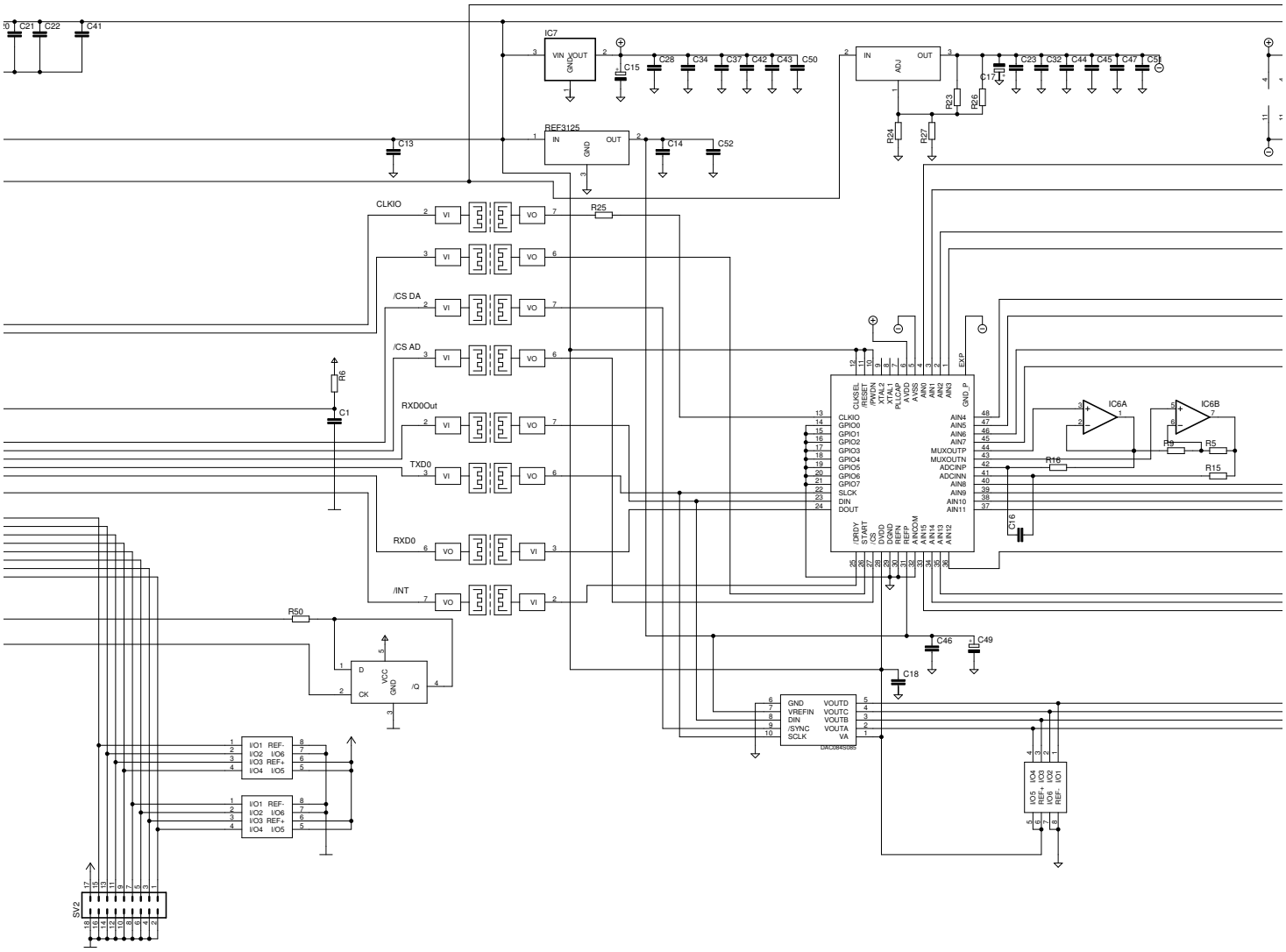
These I/O lines are connected to 100KΩ pull up resistors so that these headers can be used with jumpers to code a certain configuration which can then be read by the PC. Note that special care has to be taken if these ports are used for other purposes because they are not protected by a diode array.

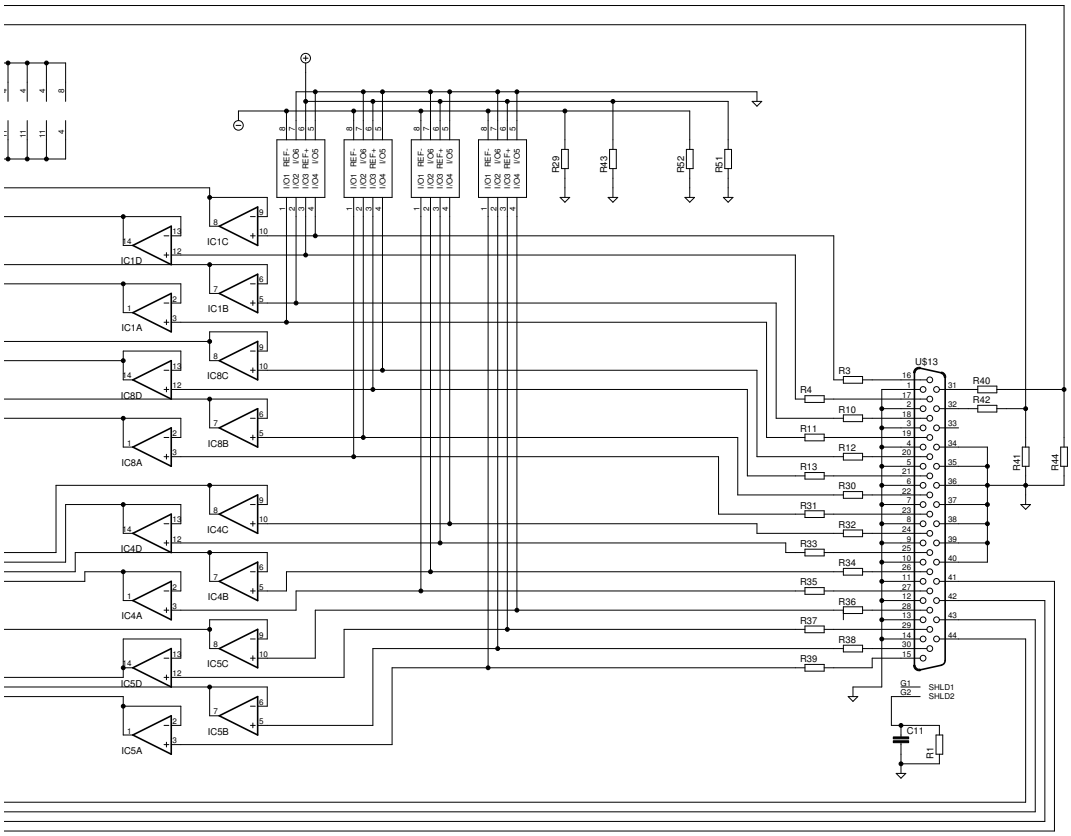


Pin	Direction	Function
2	I/O	digital I/O bit 0,4
4	I/O	digital I/O bit 1,5
6	I/O	digital I/O bit 2,6
8	I/O	digital I/O bit 3,7
1,3,5,7	GND	GND

2.2. Esquemático de la DAQ USBDUX-sigma


USBDUX
http://www.linux-usb-daq.co.uk/
Copyright (C) Bernd Porr
http://www.berndporr.me.uk
DC1
+VIN
-VIN
+VOUT
COM
-VOUT
DC/DC CONVERTER
C24
C27
C48
C40
C39
C38
C10
C7
C33
C26
C5
C6
C4
C8
C9
C30
C31
SV3
SV1
R46
R47
R48
R49
R45
R17
R18
R22
IC3
VCC
GND
SDA
SCL
WP
A2
A1
A0
Atmel EEPROM
R7
R8
C3
C2
Q2
USB
GNDA1
GNDA2
U$4
IC2
VIN
VOUT
GND
C35
C36
C25
R2
usb ground
G2
G1
X3
R19
R20
R21
R71
R72
R73
R74
R75
I/O1
I/O2
I/O3
I/O4
REF+
REF-
I/O6
I/O5
R14
CLKOUT
GND11
PD7
PD6
PD5
PD4
GND10
RES
PE6/T2EX
PE5/INT6
PE4/RXD1OUT
PE3/RXD0OUT
PE2/T2OUT
PE1/T1OUT
PE0/T0OUT
VCC10
/INT5
PD3
PD2
PD1
VCC
GND
RDY0
RDY1
RDY2
RDY3
RDY4
RDY5
AVCC
XTALOUT
XTALIN
AGND
NC1
NC2
NC3
VCC1
DPLUS
DMINUS
GND1
VCC2
GND2
INT4
T0
T1
T2
IFCLK
RESERVED
BKPT
SCL
SDA
PD0
/WAKEUP
VCC9
/RESET
CTL5
GND8
PA7
PA6
PA5
PA4
PA3
PA2
PA1
PA0
VCC7
GND7
PC7
PC6
PC5
PC4
PC3
PC2
PC1
PC0
CTL2
CTL1
CTL0
VCC6
CTL4
CTL3


C21
C22
C41
IC7
VIN
VOUT
GND
C15
C28
C34
C37
C42
C43
C50
IN
ADJ
OUT
C17
C23
C32
C44
C45
C47
C51
R23
R26
R24
R27
REF3125
C13
C14
C52
CLKIO
/CS DA
/CS AD
RXD0Out
TXD0
RXD0
/INT
VI
VO
R25
R6
C1
R50
D
CK
VCC
/Q
I/O1 REF-
I/O2 I/O6
I/O3 REF+
I/O4 I/O5
SV2
CLKSEL
/RESET
/PWDN
XTAL2
XTAL1
PLLCAP
AVDD
AVSS
AIN0
AIN1
AIN2
AIN3
EXP
GND_P
CLKIO
GPIO0
GPIO1
GPIO2
GPIO3
GPIO4
GPIO5
GPIO6
GPIO7
SLCK
DIN
DOUT
/DRDY
START
/CS
DVDD
DGND
REFN
REFP
AGNDM
AIN15
AIN14
AIN13
AIN12
AIN4
AIN5
AIN6
AIN7
MUXOUTP
MUXOUTN
ADCINP
ADCINN
AIN8
AIN9
AIN10
AIN11
IC6A
IC6B
R9
R5
R18
R15
C16
C46
C49
C18
GND
VREFIN
DIN
/SYNC
SCLK
VOUTD
VOUTC
VOUTB
VOUTA
VA
DAC8845IB85
I/O4 I/O3
REF+ I/O2
I/O5 I/O1
REF-


I/O1 REF-
I/O2 I/O6
I/O3 REF+
I/O4 I/O5
R29
R43
R52
R51
IC1C
IC1D
IC1B
IC1A
IC8C
IC8D
IC8B
IC8A
IC4C
IC4D
IC4B
IC4A
IC5C
IC5D
IC5B
IC5A
U$13
R3
R4
R10
R11
R12
R13
R30
R31
R32
R33
R34
R35
R36
R37
R38
R39
R40
R42
R41
R44
G1
G2
SHLD1
SHLD2
C11
R1

2.3. Comedi_test

Salida de la función comedi_test realizada sobre la tarjeta DAQ USBDUX-sigma, esta función nos revela las características del device y subdevices.

```
I: Comedi version: 0.7.76
I: Comedilib version: unknown =)
I: driver name: usbduxsigma
I: device name: usbduxsigma
I:
I: subdevice 0
I: testing info...
rev 1
I: subdevice type: 1 (analog input)
  number of channels: 16
  max data value: 16777215
  ranges:
    all chans: [-2.65,2.65]
I: testing insn_read...
rev 1
comedi_do_insn returned 1, good
I: testing insn_read_0...
comedi_do_insn returned 0, good
I: testing insn_read_time...
rev 1
comedi_do_insn: 3
read time: 1973 us
I: testing cmd_no_cmd...
not applicable
I: testing cmd_probe_src_mask...
rev 1
command source mask:
  start: now|int
  scan_begin: timer
  convert: now
  scan_end: count
  stop: none|count
I: testing cmd_probe_fast_1chan...
command fast 1chan:
  start: now 0
  scan_begin: timer 250000
  convert: now 0
  scan_end: count 1
  stop: count 2
I: testing cmd_read_fast_1chan...
I: testing cmd_write_fast_1chan...
not applicable
I: testing cmd_logic_bug...
rev 1
command_test returned 1, good
I: testing cmd_fifo_depth_check...
```

```
64, 64
128, 128
256, 256
512, 510
1024, 1022
2048, 2046
4096, 4096
8192, 8192
16384, 16378
32768, 32759
I: testing cmd_start_inttrig...
I: testing mmap...
0xb7739000 ok
0xb773a000 ok
0xb773b000 ok
0xb773c000 ok
0xb773d000 ok
0xb773e000 ok
0xb773f000 ok
0xb7740000 ok
0xb7741000 ok
0xb7742000 ok
compare ok
0xb7739000 segfaulted (ok)
0xb773a000 segfaulted (ok)
0xb773b000 segfaulted (ok)
0xb773c000 segfaulted (ok)
0xb773d000 segfaulted (ok)
0xb773e000 segfaulted (ok)
0xb773f000 segfaulted (ok)
0xb7740000 segfaulted (ok)
0xb7741000 segfaulted (ok)
0xb7742000 segfaulted (ok)
I: testing read_select...
I: testing bufconfig...
buffer size 65536
max buffer size 65536
setting buffer size to 4096
buffer size set to 4096
buffer size now at 4096
setting buffer size past limit, 69632
got EPERM, good
setting buffer size to max, 65536
buffer size now at 65536
I:
I: subdevice 1
I: testing info...
rev 1
I: subdevice type: 2 (analog output)
  number of channels: 4
  max data value: 255
```

```
  ranges:
    all chans: [0,2.5]
I: testing insn_read...
rev 1
comedi_do_insn returned 1, good
I: testing insn_read_0...
comedi_do_insn returned 0, good
I: testing insn_read_time...
rev 1
comedi_do_insn: 3
read time: 0 us
I: testing cmd_no_cmd...
not applicable
I: testing cmd_probe_src_mask...
rev 1
command source mask:
  start: now|int
  scan_begin: timer
  convert: now
  scan_end: count
  stop: none|count
I: testing cmd_probe_fast_1chan...
command fast 1chan:
  start: now 0
  scan_begin: timer 1000000
  convert: now 0
  scan_end: count 1
  stop: count 2
I: testing cmd_read_fast_1chan...
not applicable
I: testing cmd_write_fast_1chan...
E: comedi_internal_trigger: Resource temporarily unavailable
I: testing cmd_logic_bug...
rev 1
command_test returned 1, good
I: testing cmd_fifo_depth_check...
not applicable
I: testing cmd_start_inttrig...
not applicable
I: testing mmap...
not applicable
I: testing read_select...
not applicable
I: testing bufconfig...
buffer size 65536
max buffer size 65536
setting buffer size to 4096
buffer size set to 4096
buffer size now at 4096
setting buffer size past limit, 69632
got EPERM, good
```

```
setting buffer size to max, 65536
buffer size now at 65536
I:
I: subdevice 2
I: testing info...
rev 1
I: subdevice type: 5 (digital I/O)
  number of channels: 24
  max data value: 1
  ranges:
    all chans: [0,5]
I: testing insn_read...
rev 1
comedi_do_insn returned 1, good
I: testing insn_read_0...
comedi_do_insn returned 0, good
I: testing insn_read_time...
rev 1
comedi_do_insn: 3
read time: 739 us
I: testing cmd_no_cmd...
got EIO, good
I: testing cmd_probe_src_mask...
not applicable
I: testing cmd_probe_fast_1chan...
not applicable
I: testing cmd_read_fast_1chan...
not applicable
I: testing cmd_write_fast_1chan...
not applicable
I: testing cmd_logic_bug...
not applicable
I: testing cmd_fifo_depth_check...
not applicable
I: testing cmd_start_inttrig...
not applicable
I: testing mmap...
not applicable
I: testing read_select...
not applicable
I: testing bufconfig...
buffer length is 0, good
I:
I: subdevice 3
I: testing info...
rev 1
I: subdevice type: 12 (pwm)
  number of channels: 8
  max data value: 512
  ranges:
    all chans: [0,1]
```

I: testing insn_read...
rev 1
E: comedi_do_insn: Invalid argument
I: testing insn_read_0...
E: comedi_do_insn: Invalid argument
I: testing insn_read_time...
rev 1
comedi_do_insn: -1
W: comedi_do_insn: errno=22 Invalid argument
W: comedi_do_insn: returned -1 (expected 3)
read time: -1042937256 us
I: testing cmd_no_cmd...
got EIO, good
I: testing cmd_probe_src_mask...
not applicable
I: testing cmd_probe_fast_1chan...
not applicable
I: testing cmd_read_fast_1chan...
not applicable
I: testing cmd_write_fast_1chan...
not applicable
I: testing cmd_logic_bug...
not applicable
I: testing cmd_fifo_depth_check...
not applicable
I: testing cmd_start_inttrig...
not applicable
I: testing mmap...
not applicable
I: testing read_select...
not applicable
I: testing bufconfig...
buffer length is 0, good

2.4. Programa usados para la interactuación con la USBDUX-sigma:

2.4.1. Emisión GPOS síncrono

```
#include <stdio.h>
#include <comedilib.h>
#include <fcntl.h>
#include <unistd.h>
#include <stdlib.h>
#include <errno.h>
#include <getopt.h>
#include <ctype.h>
#include <sys/time.h>
#include <sys/wait.h>

comedi_t *device;


int main(int argc, char *argv[])
{
        lsampl_t data;
      int ch;
        int ret;
      char filename[]="/dev/comedi0";

      if (argc!=3) {
            fprintf(stderr,"hgdghh%s <chan> <value>\n",argv[0]);
            exit(-1);
      }

        device = comedi_open(filename);
      if(!device){
                comedi_perror(filename);
                exit(-1);
        }

      ch = atoi(argv[1]);
      data = atoi(argv[2]);
while (1)
{

      ret = comedi_data_write(device,1,ch,0,0, data);
        ret = comedi_data_write(device,1,ch,0,0, 0);
      //usleep(1500);

}
      return 0;
}
```

2.4.2. Adquisición GPOS síncrono

```
#include <stdio.h>   /* for printf() */
#include <comedilib.h>
#include <stdlib.h>

int subdev = 0;         /* change this to your input subdevice */
int chan = 0;           /* change this to your channel */
int range = 0;          /* more on this later */
int aref = AREF_GROUND; /* more on this later */

int main(int argc,char *argv[])
{
      comedi_t *it;
      lsampl_t data;
      int i;

      if (argc < 2) {
            fprintf(stderr,"%s channel\n",argv[0]);
            exit(0);
      }

      it=comedi_open("/dev/comedi0");

      chan = atoi(argv[1]);
      while(1){
      comedi_data_read(it,subdev,chan,range,aref, &data);

      printf("%08x  ",data);
      for(i = 0;i<data/0x080000;i++) {
            printf(" ");
      }
      printf("o\n");
}
      return 0;
}
```

2.4.3. Emisión RTAI síncrono

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <signal.h>
#include <fcntl.h>
#include <sched.h>

#include <rtai_comedi.h>

#include <rtai_lxrt.h>
//#include <rtai_sem.h>
//#include <rtai_msg.h>

#define AO_RANGE  0

static RT_TASK sound_task;      /* we'll fill this in with our task */
static RTIME sound_period_ns = 2000000; /* timer period, in nanoseconds
*/
static RTIME stop =1000000;

static int thread0;

static comedi_t *dev;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;

static int init_board(void)
{
      dev = comedi_open("/dev/comedi0");
      printf("Comedi device (6071) handle: %p.\n", dev);
      if (!dev){
            printf("Unable to open (6071) %s.\n", "/dev/comedi0");
            return 1;
      }
      //subdev = comedi_get_subdevice_type(dev, COMEDI_SUBD_A0);



      subdev = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AO,1);

      //subdev = 1;
      if (subdev < 0) {
            comedi_close(dev);
            printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AO);
            return 1;


      }

      comedi_get_krange(dev, subdev, 0, AO_RANGE, &krange);
      maxdata = comedi_get_maxdata(dev, subdev, 0);
```

```
      return 0;
}



int main(void)
{
    RT_TASK *task;
lsampl_t data;
int i;
 RTIME sound_period_count;     /* requested timer period, in counts */
  RTIME timer_period_count;     /* actual timer period, in counts */
  int retval;                   /* we look at our return values */

if (init_board()) {;
            printf("Board initialization failed.\n");
            return 1;
      }

    // make main thread LXRT soft realtime
    task  =  rt_task_init_schmod(nam2num("MYTASK"),  0,  0,  0,  SCHED_FIFO,
0xF);
    mlockall(MCL_CURRENT | MCL_FUTURE);


  /*
    Set up the timer to expire in pure periodic mode by calling

    void rt_set_periodic_mode(void);

    This sets the periodic mode for the timer. It consists of a fixed
    frequency timing of the tasks in multiple of the period set with a
    call to start_rt_timer. The resolution is that of the 8254
    frequency (1193180 hz). Any timing request not an integer multiple
    of the period is satisfied at the closest period tick. It is the
    default mode when no call is made to set the oneshot mode.
  */
//  rt_set_periodic_mode();

  /*
    Start the periodic timer by calling

    RTIME start_rt_timer(RTIME period);

    This  starts  the  timer  with  the  period  'period'  in  internal  count
units.
    It's usually convenient to provide periods in second-like units, so
    we  use  the  nano2count()  conversion  to  convert  our  period,  in
nanoseconds,
    to counts. The return value is the actual period set up, which may
    differ from the requested period due to roundoff to the allowable
    chip frequencies.

    Look at the console, or /var/log/messages, to see the printk()
    messages.
  */

rt_set_oneshot_mode();

  sound_period_count = nano2count(sound_period_ns);
```

```
  timer_period_count = start_rt_timer(sound_period_count);
  printf("periodic_task: requested %d counts, got %d counts\n",
            (int) sound_period_count, (int) timer_period_count);


 retval =
    rt_task_make_periodic(task, /* pointer to our task structure */
                            rt_get_time() + sound_period_count,
                           sound_period_count); /* recurring period */

      //rt_make_hard_real_time();
while(1){

      comedi_data_write(dev,subdev,0,0,0, 65);

      rt_busy_sleep(875000);

      comedi_data_write(dev,subdev,0,0,0, 0);

       rt_task_wait_period();
}

    // create a linux thread
    //thread0 = rt_thread_create(fun0, NULL, 10000);

    // wait for end of program
    printf("TYPE <ENTER> TO TERMINATE\n");
    getchar();

    // cleanup stuff
    stop_rt_timer();
    return 0;
}
```

2.4.4. Adquisición RTAI síncrono

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
//#include <sys/time.h>
//#include <signal.h>
#include <fcntl.h>
#include <sched.h>

#include <rtai_comedi.h>

#include <rtai_lxrt.h>
//#include <rtai_sem.h>
//#include <rtai_msg.h>

#define AI_RANGE  0

static RT_TASK sound_task;      /* we'll fill this in with our task */
static RTIME sound_period_ns = 30000000; /* timer period, in nanoseconds
*/
static int thread0;
static comedi_t *dev;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;

static int init_board(void)
{
      dev = comedi_open("/dev/comedi0");
      printf("Comedi device (6071) handle: %p.\n", dev);
      if (!dev){
            printf("Unable to open (6071) %s.\n", "/dev/comedi0");
            return 1;
      }
      subdev = comedi_get_subdevice_type(dev,COMEDI_SUBD_AI);
      //subdev = 0;
      if (subdev < 0) {
            comedi_close(dev);
            printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AI);
            return 1;
      }
      comedi_get_krange(dev, subdev, 0, AI_RANGE, &krange);
      maxdata = comedi_get_maxdata(dev, subdev, 0);

      return 0;
}



int main(void)
{
    RT_TASK *task;
```

```
lsampl_t data;
int i;

RTIME sound_period_count;     /* requested timer period, in counts */
  RTIME timer_period_count;     /* actual timer period, in counts */
  int retval;                   /* we look at our return values */

if (init_board()) {;
            printf("Board initialization failed.\n");
            return 1;
      }
    // make main thread LXRT soft realtime
    task = rt_task_init_schmod(nam2num("MYTASK2"), 10, 0, 0, SCHED_FIFO,
0);
    mlockall(MCL_CURRENT | MCL_FUTURE);


  /*
    Set up the timer to expire in pure periodic mode by calling

    void rt_set_periodic_mode(void);

    This sets the periodic mode for the timer. It consists of a fixed
    frequency timing of the tasks in multiple of the period set with a
    call to start_rt_timer. The resolution is that of the 8254
    frequency (1193180 hz). Any timing request not an integer multiple
    of the period is satisfied at the closest period tick. It is the
    default mode when no call is made to set the oneshot mode.
  */
//  rt_set_periodic_mode();

  /*
    Start the periodic timer by calling

    RTIME start_rt_timer(RTIME period);

    This starts the timer with the period 'period' in internal count
units.
    It's usually convenient to provide periods in second-like units, so
    we use the nano2count() conversion to convert our period, in
nanoseconds,
    to counts. The return value is the actual period set up, which may
    differ from the requested period due to roundoff to the allowable
    chip frequencies.

    Look at the console, or /var/log/messages, to see the printk()
    messages.
  */

rt_set_oneshot_mode();

  sound_period_count = nano2count(sound_period_ns);
  timer_period_count = start_rt_timer(sound_period_count);
  printf("periodic_task: requested %d counts, got %d counts\n",
            (int) sound_period_count, (int) timer_period_count);


 retval =
    rt_task_make_periodic(task, /* pointer to our task structure */
                            rt_get_time() + sound_period_count,
                           sound_period_count); /* recurring period */
```

L

```
while(1){
comedi_data_read(dev,0,0,0,0, &data);

      //rt_printk("%08x  ",data);
      printf("%08x  ",data);
      for(i = 0;i<data/0x080000;i++) {
      //    rt_printk(" ");
      printf(" ");
      }
      //rt_printk("o\n");
printf("o\n");

  rt_task_wait_period();
}


    // create a linux thread
    //thread0 = rt_thread_create(fun0, NULL, 10000);

    // wait for end of program
    printf("TYPE <ENTER> TO TERMINATE\n");
    getchar();



    // cleanup stuff
    stop_rt_timer();
    return 0;
}
```

2.4.5. Emisión RTAI asíncrono

```
/*
COPYRIGHT (C) 2009 Edoardo Vigoni (vigoni@aero.polimi.it)
COPYRIGHT (C) 2009 Paolo Mantegazza <mantegazza@aero.polimi.it>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/time.h>
#include <signal.h>
#include <pthread.h>
#include <math.h>


#include <rtai_comedi.h>

#define NCHAN 1

#define SIN_FREQ 20
#define N_PNT 50
#define RUN_TIME 5

#define AO_RANGE 0
#define SAMP_FREQ 1000
#define SAMP_TIME (1000000000/SAMP_FREQ)
#define TRIGSAMP 2048

static comedi_t *dev;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;

static int init_board(void)
{
dev = comedi_open("/dev/comedi0");
printf("Comedi device (6071) handle: %p.\n", dev);
if (!dev){
printf("Unable to open (6071) %s.\n", "/dev/comedi0");
return 1;
}
```

```
subdev = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AO, 0);
if (subdev < 0) {
comedi_close(dev);
printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AO);
return 1;
}
comedi_get_krange(dev, subdev, 0, AO_RANGE, &krange);
maxdata = comedi_get_maxdata(dev, subdev, 0);
return 0;
}

comedi_cmd cmd;
int do_cmd(void)
{
int ret, i;
unsigned int chanlist[NCHAN];
unsigned int buf_write[NCHAN] = { 0, 1 };

memset(&cmd, 0, sizeof(cmd));
for (i = 0; i < NCHAN; i++) {
chanlist[i] = CR_PACK(buf_write[i], AO_RANGE, AREF_GROUND);
}

cmd.subdev = subdev;
cmd.flags = TRIG_RT | TRIG_WRITE;

cmd.start_src = TRIG_INT;
cmd.start_arg = 0;

cmd.scan_begin_src = TRIG_TIMER;
cmd.scan_begin_arg = SAMP_TIME;

cmd.convert_src = TRIG_NOW;
cmd.convert_arg = 0;

cmd.scan_end_src = TRIG_COUNT;
cmd.scan_end_arg = NCHAN;

cmd.stop_src = TRIG_COUNT;
cmd.stop_arg = 10;

cmd.chanlist = chanlist;
cmd.chanlist_len = NCHAN;

ret = comedi_command_test(dev, &cmd);
ret = comedi_command_test(dev, &cmd);
printf("Comedi_command returned: %d.\n", ret);
if (ret) {
return ret;
}

ret = comedi_command(dev, &cmd);
printf("Comedi_command returned: %d.\n", ret);

return ret;
}

static volatile int end;
void endme(int sig) { end = 1; }

int main(void)
```

```
{

RTIME until;
RT_TASK *task;

lsampl_t data[SAMP_FREQ*RUN_TIME];
long k, sinewave, retval = 0;

int i, up;

signal(SIGKILL, endme);
signal(SIGTERM, endme);

start_rt_timer(0);
task = rt_task_init_schmod(nam2num("MYTASK"), 1, 0, 0, SCHED_FIFO, 0xF);
printf("COMEDI CMD TEST BEGINS: SAMPLING FREQ: %d, RUN TIME: %d.\n",
SAMP_FREQ, RUN_TIME);

if (init_board()) {;
printf("Board initialization failed.\n");
return 1;
}
i=0;
up=0;



//CASO DE MiCROEVENTO 1 milisegundo


for (k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {
up++;
if (up<8)
{
if (i==0){
      data[k] =65;
      i=1;
      }
else{
      data[k] =  0;
      i=0;

      }
}
else
{
      if (up==16)
      up=0;
      i=0;
      data[k] =  0;

}
}
for (k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++)
printf(" %d\n", data[k]);
i=0;
up=0;

//CASO DE MiCROEVENTO 1 milisegundo

/*
```

```
for (k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {

if (i==0){
      data[k] =65;
      i=1;}
else{
      data[k] =  0;

i=0;
}
//i++;
}

*/

//Caso de Microevento de duracion X=10
/*
up=5; //milisegeundos
for (k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {

if (i<(up+1))
      data[k] =127;

else{
      data[k] =  0;
      if(i==((up*2)))
      i=0;
}
i++;
}

/*
for (k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++)
printf(" %d\n", data[k]);*/

do_cmd();

mlockall(MCL_CURRENT | MCL_FUTURE);
//rt_make_hard_real_time();

until = rt_get_cpu_time_ns() + (long long)RUN_TIME*1000000000;
for (k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {

rt_comedi_command_data_write(dev, subdev, NCHAN, &data[k]);
rt_sleep(nano2count(SAMP_TIME/2));

if (k == TRIGSAMP) {
rt_comedi_trigger(dev, subdev);
}
}

while (until > rt_get_cpu_time_ns()) {
//rt_sleep(nano2count(0));
}
comedi_cancel(dev, subdev);
comedi_close(dev);
comedi_data_write(dev, subdev, 0, 0, AREF_GROUND, 2048);
comedi_data_write(dev, subdev, 1, 0, AREF_GROUND, 2048);
printf("COMEDI TEST ENDS.\n");

if (retval < 0) {
```

```
printf("rt_comedi_wait_timed overruns: %d\n", abs(retval));
}

stop_rt_timer();
rt_make_soft_real_time();
rt_task_delete(task);

return 0;
}
```

2.4.6. Adquisición RTAI asíncrono

```
/*
COPYRIGHT (C) 2009 Edoardo Vigoni (vigoni@aero.polimi.it)
COPYRIGHT (C) 2009 Paolo Mantegazza <mantegazza@aero.polimi.it>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/time.h>
#include <signal.h>
#include <pthread.h>
//#include <comedilib.h>
#include <rtai_comedi.h>
#define ONECALL 1
#define TIMEDCALL 1

#define TIMEOUT 1000000

#define NCHAN 1

#define SAMP_FREQ 1000
#define RUN_TIME 3

#define AI_RANGE 0
#define SAMP_TIME 1000000000/SAMP_FREQ
static comedi_t *dev;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;

//struct comedi_device device ;

#define BUFSZ 150
char buf[BUFSZ];


static int init_board(void)
{
dev = comedi_open("/dev/comedi0");
//device = (struct comedi_device *)dev;
```

```
printf("Comedi device (6071) handle: %p.\n", dev);
if (!dev){
printf("Unable to open (6071) %s.\n", "/dev/comedi0");
return 1;
}
subdev = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AI, 0);
if (subdev < 0) {
comedi_close(dev);
printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AI);
return 1;
}
comedi_get_krange(dev, subdev, 0, AI_RANGE, &krange);
maxdata = comedi_get_maxdata(dev, subdev, 0);
return 0;
}

int do_cmd(void)
{
int ret, i;
comedi_cmd cmd;
unsigned int chanlist[NCHAN];
unsigned int buf_read[NCHAN];

memset(&cmd, 0, sizeof(cmd));
for (i = 0; i < NCHAN; i++) {
chanlist[i] = CR_PACK(0+i, AI_RANGE, AREF_GROUND);
}

cmd.subdev = subdev;
cmd.flags = TRIG_WAKE_EOS;

cmd.start_src = TRIG_NOW;
cmd.start_arg = 0;

cmd.scan_begin_src = TRIG_TIMER;
cmd.scan_begin_arg = SAMP_TIME;

cmd.convert_src = TRIG_NOW;
cmd.convert_arg =0;

cmd.scan_end_src = TRIG_COUNT;
cmd.scan_end_arg = NCHAN;

cmd.stop_src = TRIG_NONE;
cmd.stop_arg = 0;

cmd.chanlist = chanlist;
cmd.chanlist_len = NCHAN;

ret = comedi_command_test(dev, &cmd);
printf("1st comedi_command_test returned: %d.\n", ret);
ret = comedi_command_test(dev, &cmd);
printf("2nd comedi_command_test returned: %d.\n", ret);
printf("CONVERT ARG: %d\n", cmd.convert_arg);

if (ret) {
return ret;
}

ret = comedi_command(dev, &cmd);
printf("Comedi_command returned: %d.\n", ret);
```

```
return ret;
}

static volatile int end;
void endme(int sig) { end = 1; }

int main(void)
{
RT_TASK *task;
lsampl_t data1[2];
lsampl_t d;
lsampl_t dant;
int hist[SAMP_FREQ*RUN_TIME][NCHAN];
lsampl_t data[SAMP_FREQ*RUN_TIME][NCHAN];
unsigned int val;
long i, k=0, n, cnt = 0, retval = 0;
FILE *fp;
int subdev_flags;
int bytes_per_sample;

signal(SIGKILL, endme);
signal(SIGTERM, endme);
hist = malloc(SAMP_FREQ*RUN_TIME*NCHAN*sizeof(int) + 1000);
memset(hist, 0, SAMP_FREQ*RUN_TIME*NCHAN*sizeof(int) + 1000);

start_rt_timer(0);
task = rt_task_init_schmod(nam2num("MYTASK"), 1, 0, 0, SCHED_FIFO, 0xF);
printf("COMEDI CMD TEST BEGINS: SAMPLING FREQ: %d, RUN TIME: %d.\n",
SAMP_FREQ, RUN_TIME);
mlockall(MCL_CURRENT | MCL_FUTURE);
//rt_make_hard_real_time();

if (init_board()) {;
printf("Board initialization failed.\n");
return 1;
}
rt_comedi_register_callback(dev, subdev, COMEDI_CB_EOS, NULL, task);

subdev_flags = comedi_get_subdevice_flags(dev, subdev);


do_cmd();


for (n = k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {
#if ONECALL

val = COMEDI_CB_EOS;
#if TIMEDCALL
data1[0] = data1[1] = 0;
retval = rt_comedi_command_data_wread_timed(dev, subdev, NCHAN, data1,
nano2count(TIMEOUT), &val);
retval = rt_comedi_command_data_wread_timed(dev, subdev, NCHAN, data1 +
1, nano2count(TIMEOUT), &val);
if(data1[0]!=data1[1]){
if(data1[0]<0x0000ff && dant< 0x0000ff)
d=(data1[0] << 16) + data1[1];
else
d=(data1[1] << 16) + data1[0];
printf("%d       %x    %x \n", d, data1[0],data1[1]);
```

```
dant=data1[0];
}

#else
retval += rt_comedi_command_data_wread(dev, subdev, NCHAN, data,&val);
#endif

#else


#if TIMEDCALL
retval += rt_comedi_wait_timed(nano2count(TIMEOUT), &val);
#else
retval += rt_comedi_wait(&val);
#endif

#endif
if (val & COMEDI_CB_EOS) {
#if !ONECALL
rt_comedi_command_data_read(dev, subdev, NCHAN, data);

else

rt_sleep(nano2count(SAMP_TIME/10));

#endif
} else {

rt_sleep(nano2count(SAMP_TIME/10));

}
}
comedi_cancel(dev, subdev);
comedi_close(dev);
printf("COMEDI TEST ENDS.\n");

if (retval < 0) {
printf("rt_comedi_wait_timed overruns: %d\n", abs(retval));
}
fp = fopen("rec.dat", "w");
if(subdev_flags & SDF_LSAMPL)
                        bytes_per_sample = sizeof(lsampl_t);

                  else
                        bytes_per_sample = sizeof(sampl_t);

fprintf(fp, "bytes_per_sample %d: ",bytes_per_sample );
//for (n = k = 0; k < SAMP_FREQ*RUN_TIME; k++) {
/*for (n = k = 0; k < SAMP_FREQ*RUN_TIME; k++) {
fprintf(fp, "# %u: ", k);
for (i = 0; i < NCHAN; i++) {
if(subdev_flags & SDF_LSAMPL) {
                              hist[k][i]=((lsampl_t )data[k][i]);
                        } else {
                              hist[k][i]= ((sampl_t *)data[k][i]);
                        }

fprintf(fp, "%x\t", hist[k][i]);
}
fprintf(fp, "\n");
}
```

```
fclose(fp);
free(hist);*/
stop_rt_timer();
rt_make_soft_real_time();
rt_task_delete(task);

return 0;
}
```

2.4.7. Ciclo cerrado configuración síncrona.

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <string.h>
#include <sys/types.h>
#include <sys/mman.h>
#include <sys/stat.h>
#include <sys/time.h>
#include <signal.h>
#include <fcntl.h>
#include <sched.h>

#include <rtai_comedi.h>

#include <rtai_lxrt.h>
//#include <rtai_sem.h>
//#include <rtai_msg.h>

#define AO_RANGE  0

static RT_TASK sound_task;      /* we'll fill this in with our task */
static RTIME sound_period_ns = 10000000; /* timer period, in nanoseconds
*/



static comedi_t *dev;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;


static int control;
static int contador;


static lsampl_t data;

static lsampl_t data1;

static lsampl_t data2;


static int init_board(void)
{
      dev = comedi_open("/dev/comedi0");
      printf("Comedi device (6071) handle: %p.\n", dev);
      if (!dev){
            printf("Unable to open (6071) %s.\n", "/dev/comedi0");
            return 1;
      }
      //subdev = comedi_get_subdevice_type(dev, 0);
      subdev = 1;
      if (subdev < 0) {
            comedi_close(dev);
            printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AO);
            return 1;
      }
```

```
      comedi_get_krange(dev, subdev, 0, AO_RANGE, &krange);
      maxdata = comedi_get_maxdata(dev, subdev, 0);

      return 0;
}


int main(void)
{
    RT_TASK *task;


int i;
control=0;
lsampl_t d;

RTIME sound_period_count;     /* requested timer period, in counts */
  RTIME timer_period_count;     /* actual timer period, in counts */
  int retval;                   /* we look at our return values */

if (init_board()) {;
            printf("Board initialization failed.\n");
            return 1;
      }


    // make main thread LXRT soft realtime
    task = rt_task_init_schmod(nam2num("MYTASK"), 0, 0, 0, SCHED_FIFO,
0xF);
    mlockall(MCL_CURRENT | MCL_FUTURE);


  /*
    Set up the timer to expire in pure periodic mode by calling

    void rt_set_periodic_mode(void);

    This sets the periodic mode for the timer. It consists of a fixed
    frequency timing of the tasks in multiple of the period set with a
    call to start_rt_timer. The resolution is that of the 8254
    frequency (1193180 hz). Any timing request not an integer multiple
    of the period is satisfied at the closest period tick. It is the
    default mode when no call is made to set the oneshot mode.
  */
//  rt_set_periodic_mode();

  /*
    Start the periodic timer by calling

    RTIME start_rt_timer(RTIME period);

    This starts the timer with the period 'period' in internal count
units.
    It's usually convenient to provide periods in second-like units, so
    we use the nano2count() conversion to convert our period, in
nanoseconds,
    to counts. The return value is the actual period set up, which may
    differ from the requested period due to roundoff to the allowable
    chip frequencies.
```

```
    Look at the console, or /var/log/messages, to see the printk()
    messages.
  */

rt_set_oneshot_mode();

  sound_period_count = nano2count(sound_period_ns);
  timer_period_count = start_rt_timer(sound_period_count);
  printf("periodic_task: requested %d counts, got %d counts\n",
           (int) sound_period_count, (int) timer_period_count);
 retval =
    rt_task_make_periodic(task, /* pointer to our task structure */
                          rt_get_time() + sound_period_count,
                         sound_period_count); /* recurring period */


data1=0x000000;
data2=0x000000;
while(1){



comedi_data_read(dev,0,0,0,0, &d);

data2=data1;
data1=data;
data=d;

//rt_printk("%08x  ",d);
      printf("%08x  ",data);
      for(i = 0;i<d/0x080000;i++) {
            //rt_printk(" ");
      printf(" ");
      }
      //rt_printk("o\n");
printf("o\n");

if(data>data1 && data1<data2 ){

if (contador==2){

comedi_data_write(dev,1,0,0,0, 65);
rt_busy_sleep(1000000);

comedi_data_write(dev,1,0,0,0, 0);

contador=0;
}

contador=contador++;

}

  rt_task_wait_period();
}

    // start realtime timer and scheduler
 /*   rt_set_oneshot_mode();
    start_rt_timer(0);
*/
    // create a linux thread
```

```
    //thread0 = rt_thread_create(fun0, NULL, 10000);

    // wait for end of program
    printf("TYPE <ENTER> TO TERMINATE\n");
    getchar();



    // cleanup stuff
    stop_rt_timer();
    return 0;
}
```

2.4.8. Ciclo cerrado configuración mixta.

```
/*
COPYRIGHT (C) 2009 Edoardo Vigoni (vigoni@aero.polimi.it)
COPYRIGHT (C) 2009 Paolo Mantegazza <mantegazza@aero.polimi.it>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/time.h>
#include <signal.h>
#include <pthread.h>
//#include <comedilib.h>
#include <rtai_comedi.h>
#define ONECALL 1
#define TIMEDCALL 1

#define TIMEOUT 1000000

#define NCHAN 1
#define NCHAN_AO 1

#define SAMP_FREQ 500
#define RUN_TIME 60

#define AI_RANGE 0
#define SAMP_TIME 1000000000/SAMP_FREQ
#define SAMP_TIME_AO 1000000000/SAMP_FREQ_AO
#define AO_RANGE 0
#define SAMP_FREQ_AO 1000

static comedi_t *dev;
static comedi_t *dev_ao;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;
static int subdev_ao;
static comedi_krange krange_ao;
static lsampl_t maxdata_ao;


//struct comedi_device device ;
```

```
#define BUFSZ 150
char buf[BUFSZ];


static int init_board(void)
{
dev = comedi_open("/dev/comedi0");
dev_ao = comedi_open("/dev/comedi0");
//device = (struct comedi_device *)dev;
printf("Comedi device (6071) handle: %p.\n", dev);
if (!dev){
printf("Unable to open (6071) %s.\n", "/dev/comedi0");
return 1;
}
printf("Comedi device (6071) handle: %p.\n", dev_ao);
if (!dev_ao){
printf("Unable to opensdgg (6071) %s.\n", "/dev/comedi0");
return 1;
}
subdev = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AI, 0);
if (subdev < 0) {
comedi_close(dev);
printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AI);
return 1;
}

subdev_ao = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AO, 0);
if (subdev_ao < 0) {
comedi_close(dev);
printf("subdev_ao (6071) %d not found.\n", COMEDI_SUBD_AO);
return 1;
}

comedi_get_krange(dev, subdev_ao, 0, AO_RANGE, &krange_ao);
maxdata_ao = comedi_get_maxdata(dev, subdev_ao, 0);
comedi_get_krange(dev, subdev, 0, AI_RANGE, &krange);
maxdata = comedi_get_maxdata(dev, subdev, 0);


int do_cmd(void)
{
int ret, i;
comedi_cmd cmd;
unsigned int chanlist[NCHAN];
unsigned int buf_read[NCHAN]={0,1};

memset(&cmd, 0, sizeof(cmd));
for (i = 0; i < NCHAN; i++) {
chanlist[i] = CR_PACK(buf_read[i], AI_RANGE, AREF_GROUND);
}

cmd.subdev = subdev;
cmd.flags = TRIG_WAKE_EOS;

cmd.start_src = TRIG_NOW;
cmd.start_arg = 0;

cmd.scan_begin_src = TRIG_TIMER;
cmd.scan_begin_arg = SAMP_TIME;
```

```
cmd.convert_src = TRIG_NOW;
cmd.convert_arg =0;

cmd.scan_end_src = TRIG_COUNT;
cmd.scan_end_arg = NCHAN;

cmd.stop_src = TRIG_NONE;
cmd.stop_arg = 0;

cmd.chanlist = chanlist;
cmd.chanlist_len = NCHAN;

ret = comedi_command_test(dev, &cmd);
printf("1st comedi_command_test returned OUTPUT: %d.\n", ret);
ret = comedi_command_test(dev, &cmd);
printf("2nd comedi_command_test returned: %d.\n", ret);
printf("CONVERT ARG: %d\n", cmd.convert_arg);

if (ret) {
return ret;
}

ret = comedi_command(dev, &cmd);
printf("Comedi_command returned: %d.\n", ret);

return ret;
}

static volatile int end;
void endme(int sig) { end = 1; }

int main(void)
{
int h=0;
lsampl_t emision[10] ={16,26,36,46,56,66,56,46,36,26};
RT_TASK *task;
lsampl_t d[2];
lsampl_t data1;
lsampl_t data2;
lsampl_t dant;
//int hist[SAMP_FREQ*RUN_TIME][NCHAN];
lsampl_t data3[9];
lsampl_t data;
unsigned int val;
long i, k=0, n, cnt = 0, retval = 0;
//FILE *fp;
int subdev_flags;
//int bytes_per_sample;
int contador;
int up;

RTIME until;
signal(SIGKILL, endme);
signal(SIGTERM, endme);

start_rt_timer(0);
task = rt_task_init_schmod(nam2num("MYTASK"), 1, 0, 0, SCHED_FIFO, 0xF);
printf("COMEDI CMD TEST BEGINS: SAMPLING FREQ: %d, RUN TIME: %d.\n",
SAMP_FREQ, RUN_TIME);
mlockall(MCL_CURRENT | MCL_FUTURE);
//rt_make_hard_real_time();
```

```
if (init_board()) {;
printf("Board initialization failed.\n");
return 1;
}
rt_comedi_register_callback(dev, subdev, COMEDI_CB_EOS, NULL, task);

subdev_flags = comedi_get_subdevice_flags(dev, subdev);


i=0;
up=0;

do_cmd();
data1=0x000000;
data2=0x000000;
contador =0;
until = rt_get_cpu_time_ns() + (long long)RUN_TIME*1000000000;
for (n = k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {

retval += rt_comedi_wait_timed(nano2count(TIMEOUT), &val);

if (val & COMEDI_CB_EOS) {

      rt_comedi_command_data_read(dev, subdev, NCHAN, d);
      rt_comedi_command_data_read(dev, subdev, NCHAN, d +1);
      //printf("%08x  %08x  %08x\n",(d[0] << 16) + d[1],d[1],d[0] );
      data2=data1;
      data1=data;
      if(d[0]<0x0000ff && dant<0x0000ff)
      data=(d[0] << 16) ;//+ d[1];
      else
      data=(d[1] << 16) ;//+ d[0];
      up=0;
      dant=d[0];

      if(data<data1 && data1>data2 ){
            //&& data> 0X0990000&& data> 0X00600000
            comedi_data_write(dev,1,0,0,AREF_GROUND,65);
            comedi_data_write(dev,1,0,0,AREF_GROUND,0);
      //    comedi_data_write(dev,1,0,0,AREF_GROUND,65);
      //    comedi_data_write(dev,1,0,0,AREF_GROUND,0);
            //rt_sleep(nano2count(SAMP_TIME/2));
      }else{
            //rt_sleep(nano2count(SAMP_TIME/10));///10
            comedi_data_write(dev,1,0,0,AREF_GROUND,0);
            comedi_data_write(dev,1,0,0,AREF_GROUND,0);
            //comedi_data_write(dev,1,0,0,AREF_GROUND,0);
            //comedi_data_write(dev,1,0,0,AREF_GROUND,0);

      }
      ///rt_comedi_command_data_write(dev,      subdev_ao,      NCHAN_AO,
&emision[3]);
} else {

            //if(data>data1 && data1<data2 && data> 0X00600000 ){

}

}
```

```
comedi_cancel(dev, subdev);
comedi_cancel(dev, subdev_ao);
comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 2048);
comedi_close(dev);
printf("COMEDI TEST ENDS.\n");


stop_rt_timer();
rt_make_soft_real_time();
rt_task_delete(task);

return 0;
}
```

2.4.9. Ciclo cerrado configuración asíncrona.

```
COPYRIGHT (C) 2009 Edoardo Vigoni (vigoni@aero.polimi.it)
COPYRIGHT (C) 2009 Paolo Mantegazza <mantegazza@aero.polimi.it>

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA.
*/


#include <stdio.h>
#include <unistd.h>
#include <string.h>
#include <signal.h>
#include <sys/mman.h>
#include <sys/types.h>
#include <sys/time.h>
#include <signal.h>
#include <pthread.h>
//#include <comedilib.h>
#include <rtai_comedi.h>
#define ONECALL 1
#define TIMEDCALL 1

#define TIMEOUT 1000000

#define NCHAN 1
#define NCHAN_AO 1

#define SAMP_FREQ 1000
#define RUN_TIME 60

#define AI_RANGE 0
#define SAMP_TIME 1000000000/SAMP_FREQ
#define SAMP_TIME_AO 1000000000/SAMP_FREQ_AO
#define AO_RANGE 0
#define SAMP_FREQ_AO 1000

static comedi_t *dev;
static comedi_t *dev_ao;
static int subdev;
static comedi_krange krange;
static lsampl_t maxdata;
static int subdev_ao;
static comedi_krange krange_ao;
static lsampl_t maxdata_ao;


//struct comedi_device device ;
```

```
#define BUFSZ 150
char buf[BUFSZ];


static int init_board(void)
{
dev = comedi_open("/dev/comedi0");
dev_ao = comedi_open("/dev/comedi0");
//device = (struct comedi_device *)dev;
printf("Comedi device (6071) handle: %p.\n", dev);
if (!dev){
printf("Unable to open (6071) %s.\n", "/dev/comedi0");
return 1;
}
printf("Comedi device (6071) handle: %p.\n", dev_ao);
if (!dev_ao){
printf("Unable to opensdgg (6071) %s.\n", "/dev/comedi0");
return 1;
}
subdev = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AI, 0);
if (subdev < 0) {
comedi_close(dev);
printf("Subdev (6071) %d not found.\n", COMEDI_SUBD_AI);
return 1;
}

subdev_ao = comedi_find_subdevice_by_type(dev, COMEDI_SUBD_AO, 0);
if (subdev_ao < 0) {
comedi_close(dev);
printf("subdev_ao (6071) %d not found.\n", COMEDI_SUBD_AO);
return 1;
}

comedi_get_krange(dev, subdev_ao, 0, AO_RANGE, &krange_ao);
maxdata_ao = comedi_get_maxdata(dev, subdev_ao, 0);
comedi_get_krange(dev, subdev, 0, AI_RANGE, &krange);
maxdata = comedi_get_maxdata(dev, subdev, 0);



return 0;
}

comedi_cmd cmd_ao;
int do_cmd_ao(void)
{
int ret, i;
unsigned int chanlist[NCHAN_AO];
unsigned int buf_write[NCHAN_AO] = { 0};

memset(&cmd_ao, 0, sizeof(cmd_ao));
for (i = 0; i < NCHAN_AO; i++) {
chanlist[i] = CR_PACK(buf_write[i], AO_RANGE, AREF_GROUND);
}

cmd_ao.subdev = subdev_ao;
cmd_ao.flags = TRIG_RT | TRIG_WRITE;

cmd_ao.start_src = TRIG_INT;
cmd_ao.start_arg = 0;
```

```
cmd_ao.scan_begin_src = TRIG_TIMER;
cmd_ao.scan_begin_arg = SAMP_TIME_AO;

cmd_ao.convert_src = TRIG_NOW;
cmd_ao.convert_arg = 0;

cmd_ao.scan_end_src = TRIG_COUNT;
cmd_ao.scan_end_arg = NCHAN_AO;

cmd_ao.stop_src = TRIG_NONE;
cmd_ao.stop_arg = 0;

cmd_ao.chanlist = chanlist;
cmd_ao.chanlist_len = NCHAN_AO;

ret = comedi_command_test(dev, &cmd_ao);
ret = comedi_command_test(dev, &cmd_ao);
printf("Comedi_command returned: %d.\n", ret);
if (ret) {
return ret;
}

ret = comedi_command(dev, &cmd_ao);
printf("Comedi_command returned: %d.\n", ret);

return ret;
}
```

```
int do_cmd(void)
{
int ret, i;
comedi_cmd cmd;
unsigned int chanlist[NCHAN];
unsigned int buf_read[NCHAN]={0,1};

memset(&cmd, 0, sizeof(cmd));
for (i = 0; i < NCHAN; i++) {
chanlist[i] = CR_PACK(buf_read[i], AI_RANGE, AREF_GROUND);
}

cmd.subdev = subdev;
cmd.flags = TRIG_WAKE_EOS;

cmd.start_src = TRIG_NOW;
cmd.start_arg = 0;

cmd.scan_begin_src = TRIG_TIMER;
cmd.scan_begin_arg = SAMP_TIME;

cmd.convert_src = TRIG_NOW;
cmd.convert_arg =0;

cmd.scan_end_src = TRIG_COUNT;
cmd.scan_end_arg = NCHAN;

cmd.stop_src = TRIG_NONE;
cmd.stop_arg = 0;
```

```
cmd.chanlist = chanlist;
cmd.chanlist_len = NCHAN;

ret = comedi_command_test(dev, &cmd);
printf("1st comedi_command_test returned OUTPUT: %d.\n", ret);
ret = comedi_command_test(dev, &cmd);
printf("2nd comedi_command_test returned: %d.\n", ret);
printf("CONVERT ARG: %d\n", cmd.convert_arg);

if (ret) {
return ret;
}

ret = comedi_command(dev, &cmd);
printf("Comedi_command returned: %d.\n", ret);

return ret;
}

static volatile int end;
void endme(int sig) { end = 1; }

int main(void)
{
int h=0;
lsampl_t emision[10] ={66,66,66,66,66,66,66,0,0 ,0};
RT_TASK *task;
lsampl_t d[2];
lsampl_t data1;
lsampl_t data2;
lsampl_t datant;
lsampl_t data;
unsigned int val;
long i, k=0, n, cnt = 0, retval = 0;
//FILE *fp;
int subdev_flags;
int up;

RTIME until;
signal(SIGKILL, endme);
signal(SIGTERM, endme);


start_rt_timer(0);
task = rt_task_init_schmod(nam2num("MYTASK"), 1, 0, 0, SCHED_FIFO, 0xF);
printf("COMEDI CMD TEST BEGINS: SAMPLING FREQ: %d, RUN TIME: %d.\n",
SAMP_FREQ, RUN_TIME);
mlockall(MCL_CURRENT | MCL_FUTURE);
//rt_make_hard_real_time();

if (init_board()) {;
printf("Board initialization failed.\n");
return 1;
}
rt_comedi_register_callback(dev, subdev, COMEDI_CB_EOS, NULL, task);

subdev_flags = comedi_get_subdevice_flags(dev, subdev);

for (k = 0; k < 9 ; k++) {
```

```
if (i==0){
    data3[k] =65;
    i=1;}
else{
    data3[k] =  0;

i=0;
}
}

for (k = 0; k < SAMP_FREQ_AO*RUN_TIME && !end; k++)


do_cmd_ao();
do_cmd();
data1=0x000000;
data2=0x000000;

until = rt_get_cpu_time_ns() + (longlong)RUN_TIME*1000000000;
for (n = k = 0; k < SAMP_FREQ*RUN_TIME && !end; k++) {

retval += rt_comedi_wait_timed(nano2count(TIMEOUT), &val);

//emision continua
if(k ==1)
 rt_comedi_trigger(dev, subdev_ao);

if (val & COMEDI_CB_EOS) {

    rt_comedi_command_data_read(dev, subdev, NCHAN, d);
    rt_comedi_command_data_read(dev, subdev, NCHAN, d +1);
    //printf("%08x  %08x  %08x\n",(d[0] << 16) + d[1],d[1],d[0] );
    data2=data1;
    data1=data;

    if(d[0]!=d[1]){
    if(d[0]<0x0000150 && datant<0x0000150)
    data=(d[0] << 16) + d[1];
    else
    data=(d[1] << 16) + d[0];
    }
    datant=d[0];


    //printf("%08x \n ",data);
    if(data<data1 && data1>data2 && data> 0X0890000){//&& data> 0X0990000
    //printf("IMPULSO \n");

    rt_comedi_command_data_write(dev, subdev_ao, NCHAN_AO, &emision[1]);
    //rt_sleep(nano2count(SAMP_TIME/2));//


    }else{

    //comedi_data_write(dev,1,0,0,AREF_GROUND,0);
    rt_comedi_command_data_write(dev, subdev_ao, NCHAN_AO, &emision[8]);
    //rt_sleep(nano2count(SAMP_TIME/2));///10
    }
}
```

```
/*
printf("%08x  ",data);
    for(i = 0;i<data/0x080000;i++) {
    //    rt_printk(" ");
    printf(" ");
    }
    //rt_printk("o\n");

printf("o\n");
*/
/*
if(data<data1 && data1>data2 ){
//comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 65);&& data>
0X0a00000
//comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 65);
//comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 65);
//comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 0);

//for(h=0;h<2;h++){
rt_comedi_command_data_write(dev, subdev_ao, NCHAN_AO, &emision[1]);
//printf("IMPULSO: %d \n",(d[1] << 16) + d[0]);
//rt_sleep(nano2count(SAMP_TIME_AO/2));
if(k>=1)
rt_comedi_trigger(dev, subdev_ao);
comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 0);
//comedi_cancel(dev,subdev_ao);
//}

//printf("IMPULSO: %x \n",data);
}*/

}

//}

while (until > rt_get_cpu_time_ns()) {
rt_sleep(nano2count(100000));
}

comedi_cancel(dev, subdev);
comedi_cancel(dev, subdev_ao);
comedi_data_write(dev, subdev_ao, 0, 0, AREF_GROUND, 2048);
comedi_close(dev);
printf("COMEDI TEST ENDS.\n");


stop_rt_timer();
rt_make_soft_real_time();
rt_task_delete(task);

return 0;
}
```

PRESUPUESTO

1) Ejecución Material

- Compra de DAQ USBDUX-sigma(Software libre)..... ..... 344,98€*
- Compra de DAQ USBDUX-fast(software libre)..... 261,71€*
- Material para realización de conexiones ..... Total (96.34€)
  - Cables con funda sonido flexib ..... 11.2 €
  - Conectores banana 2mm ..... 12.8 €
  - Conector sub-D HD 44p ..... 4.84 €
  - Conectores BNC ..... 15 €
  - Carcasa Sub-25pines presión ..... 5.8 €
  - Termorretractil ..... 3 €
  - Soldador JBC lápiz 230V ..... 28.3 €
  - Componentes pasivos ..... 3 €
  - Carrete de estaño 0.81 mm ..... 12.40 €

*precio en euros. El producto se comercializa en libras esterlinas £, cambio 1£ a 1,19€

- Total de ejecución material..... 703.03 €

2) Gastos generales

- 16 % sobre Ejecución Material..... 112.48€

3) Beneficio Industrial

- 6 % sobre Ejecución Material..... 42.18€

4) Honorarios Proyecto

- 640 horas a 15 € / hora..... 9600 €

5) Material fungible

- Gastos de impresión..... 60 €
- Encuadernación..... 200 €

6) Subtotal del presupuesto

- Subtotal Presupuesto..... 10717.7 €

7) I.V.A. aplicable

- 21% Subtotal Presupuesto ..... 2250,7 €

8) Total presupuesto

- Total Presupuesto..... 12968,5 €

Madrid, Diciembre de 2013

El Ingeniero Jefe de Proyecto
Fdo.: David Mancebo Calle
Ingeniero Superior de Telecomunicación

PLIEGO DE CONDICIONES

Este documento contiene las condiciones legales que guiarán la realización, en este proyecto, Registro en tiempo real de señalización biológica utilizando una tarjeta de adquisición de datos USB. En lo que sigue, se supondrá que el proyecto ha sido encargado por una empresa cliente a una empresa consultora con la finalidad de realizar dicho sistema. Dicha empresa ha debido desarrollar una línea de investigación con objeto de elaborar el proyecto. Esta línea de investigación, junto con el posterior desarrollo de los programas está amparada por las condiciones particulares del siguiente pliego.

Supuesto que la utilización industrial de los métodos recogidos en el presente proyecto ha sido decidida por parte de la empresa cliente o de otras, la obra a realizar se regulará por las siguientes:

Condiciones generales

1. La modalidad de contratación será el concurso. La adjudicación se hará, por tanto, a la proposición más favorable sin atender exclusivamente al valor económico, dependiendo de las mayores garantías ofrecidas. La empresa que somete el proyecto a concurso se reserva el derecho a declararlo desierto.

2. El montaje y mecanización completa de los equipos que intervengan será realizado totalmente por la empresa licitadora.

3. En la oferta, se hará constar el precio total por el que se compromete a realizar la obra y el tanto por ciento de baja que supone este precio en relación con un importe límite si este se hubiera fijado.

4. La obra se realizará bajo la dirección técnica de un Ingeniero Superior de Telecomunicación, auxiliado por el número de Ingenieros Técnicos y Programadores que se estime preciso para el desarrollo de la misma.

5. Aparte del Ingeniero Director, el contratista tendrá derecho a contratar al resto del personal, pudiendo ceder esta prerrogativa a favor del Ingeniero Director, quien no estará obligado a aceptarla.

6. El contratista tiene derecho a sacar copias a su costa de los planos, pliego de condiciones y presupuestos. El Ingeniero autor del proyecto autorizará con su firma las copias solicitadas por el contratista después de confrontarlas.

7. Se abonará al contratista la obra que realmente ejecute con sujeción al proyecto que sirvió de base para la contratación, a las modificaciones autorizadas por la superioridad o a las órdenes que con arreglo a sus facultades le hayan comunicado por escrito al Ingeniero Director de obras siempre que dicha obra se haya ajustado a los preceptos de los pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la

pliegos de condiciones, con arreglo a los cuales, se harán las modificaciones y la valoración de las diversas unidades sin que el importe total pueda exceder de los presupuestos aprobados. Por consiguiente, el número de unidades que se consignan en el proyecto o en el presupuesto, no podrá servirle de fundamento para entablar reclamaciones de ninguna clase, salvo en los casos de rescisión.

8. Tanto en las certificaciones de obras como en la liquidación final, se abonarán los trabajos realizados por el contratista a los precios de ejecución material que figuran en el presupuesto para cada unidad de la obra.

9. Si excepcionalmente se hubiera ejecutado algún trabajo que no se ajustase a las condiciones de la contrata pero que sin embargo es admisible a juicio del Ingeniero Director de obras, se dará conocimiento a la Dirección, proponiendo a la vez la rebaja de precios que el Ingeniero estime justa y si la Dirección resolviera aceptar la obra, quedará el contratista obligado a conformarse con la rebaja acordada.

10. Cuando se juzgue necesario emplear materiales o ejecutar obras que no figuren en el presupuesto de la contrata, se evaluará su importe a los precios asignados a otras obras o materiales análogos si los hubiere y cuando no, se discutirán entre el Ingeniero Director y el contratista, sometiéndolos a la aprobación de la Dirección. Los nuevos precios convenidos por uno u otro procedimiento, se sujetarán siempre al establecido en el punto anterior.

11. Cuando el contratista, con autorización del Ingeniero Director de obras, emplee materiales de calidad más elevada o de mayores dimensiones de lo estipulado en el proyecto, o sustituya una clase de fabricación por otra que tenga asignado mayor precio o ejecute con mayores dimensiones cualquier otra parte de las obras, o en general, introduzca en ellas cualquier modificación que sea beneficiosa a juicio del Ingeniero Director de obras, no tendrá derecho sin embargo, sino a lo que le correspondería si hubiera realizado la obra con estricta sujeción a lo proyectado y contratado.

12. Las cantidades calculadas para obras accesorias, aunque figuren por partida alzada en el presupuesto final (general), no serán abonadas sino a los precios de la contrata, según las condiciones de la misma y los proyectos particulares que para ellas se formen, o en su defecto, por lo que resulte de su medición final.

13. El contratista queda obligado a abonar al Ingeniero autor del proyecto y director de obras así como a los Ingenieros Técnicos, el importe de sus respectivos honorarios facultativos por formación del proyecto, dirección técnica y administración en su caso, con arreglo a las tarifas y honorarios vigentes.

14. Concluida la ejecución de la obra, será reconocida por el Ingeniero Director que a tal efecto designe la empresa.

15. La garantía definitiva será del 4% del presupuesto y la provisional del 2%.

16. La forma de pago será por certificaciones mensuales de la obra ejecutada, de acuerdo con los precios del presupuesto, deducida la baja si la hubiera.

17. La fecha de comienzo de las obras será a partir de los 15 días naturales del replanteo oficial de las mismas y la definitiva, al año de haber ejecutado la provisional, procediéndose si no existe reclamación alguna, a la reclamación de la fianza.

18. Si el contratista al efectuar el replanteo, observase algún error en el proyecto, deberá comunicarlo en el plazo de quince días al Ingeniero Director de obras, pues transcurrido ese plazo será responsable de la exactitud del proyecto.

19. El contratista está obligado a designar una persona responsable que se entenderá con el Ingeniero Director de obras, o con el delegado que éste designe, para todo relacionado con ella. Al ser el Ingeniero Director de obras el que interpreta el proyecto, el contratista deberá consultarle cualquier duda que surja en su realización.

20. Durante la realización de la obra, se girarán visitas de inspección por personal facultativo de la empresa cliente, para hacer las comprobaciones que se crean oportunas. Es obligación del contratista, la conservación de la obra ya ejecutada hasta la recepción de la misma, por lo que el deterioro parcial o total de ella, aunque sea por agentes atmosféricos u otras causas, deberá ser reparado o reconstruido por su cuenta.

21. El contratista, deberá realizar la obra en el plazo mencionado a partir de la fecha del contrato, incurriendo en multa, por retraso de la ejecución siempre que éste no sea debido a causas de fuerza mayor. A la terminación de la obra, se hará una recepción provisional previo reconocimiento y examen por la dirección técnica, el depositario de efectos, el interventor y el jefe de servicio o un representante, estampando su conformidad el contratista.

22. Hecha la recepción provisional, se certificará al contratista el resto de la obra, reservándose la administración el importe de los gastos de conservación de la misma hasta su recepción definitiva y la fianza durante el tiempo señalado como plazo de garantía. La recepción definitiva se hará en las mismas condiciones que la provisional, extendiéndose el acta correspondiente. El Director Técnico propondrá a la Junta Económica la devolución de la fianza al contratista de acuerdo con las condiciones económicas legales establecidas.

23. Las tarifas para la determinación de honorarios, reguladas por orden de la Presidencia del Gobierno el 19 de Octubre de 1961, se aplicarán sobre el denominado en la actualidad “Presupuesto de Ejecución de Contrata” y anteriormente llamado ”Presupuesto de Ejecución Material” que hoy designa otro concepto.

Condiciones particulares

La empresa consultora, que ha desarrollado el presente proyecto, lo entregará a la empresa cliente bajo las condiciones generales ya formuladas, debiendo añadirse las siguientes condiciones particulares:

1. La propiedad intelectual de los procesos descritos y analizados en el presente trabajo, pertenece por entero a la empresa consultora representada por el Ingeniero Director del Proyecto.

2. La empresa consultora se reserva el derecho a la utilización total o parcial de los resultados de la investigación realizada para desarrollar el siguiente proyecto, bien para su publicación o bien para su uso en trabajos o proyectos posteriores, para la misma empresa cliente o para otra.

3. Cualquier tipo de reproducción aparte de las reseñadas en las condiciones generales, bien sea para uso particular de la empresa cliente, o para cualquier otra aplicación, contará con autorización expresa y por escrito del Ingeniero Director del Proyecto, que actuará en representación de la empresa consultora.

4. En la autorización se ha de hacer constar la aplicación a que se destinan sus reproducciones así como su cantidad.

5. En todas las reproducciones se indicará su procedencia, explicitando el nombre del proyecto, nombre del Ingeniero Director y de la empresa consultora.

6. Si el proyecto pasa la etapa de desarrollo, cualquier modificación que se realice sobre él, deberá ser notificada al Ingeniero Director del Proyecto y a criterio de éste, la empresa consultora decidirá aceptar o no la modificación propuesta.

7. Si la modificación se acepta, la empresa consultora se hará responsable al mismo nivel que el proyecto inicial del que resulta el añadirla.

8. Si la modificación no es aceptada, por el contrario, la empresa consultora declinará toda responsabilidad que se derive de la aplicación o influencia de la misma.

9. Si la empresa cliente decide desarrollar industrialmente uno o varios productos en los que resulte parcial o totalmente aplicable el estudio de este proyecto, deberá comunicarlo a la empresa consultora.

10. La empresa consultora no se responsabiliza de los efectos laterales que se puedan producir en el momento en que se utilice la herramienta objeto del presente proyecto para la realización de otras aplicaciones.

11. La empresa consultora tendrá prioridad respecto a otras en la elaboración de los proyectos auxiliares que fuese necesario desarrollar para dicha aplicación industrial, siempre que no haga explícita renuncia a este hecho. En este caso, deberá autorizar expresamente los proyectos presentados por otros.

12. El Ingeniero Director del presente proyecto, será el responsable de la dirección de la aplicación industrial siempre que la empresa consultora lo estime oportuno. En caso contrario, la persona designada deberá contar con la autorización del mismo, quien delegará en él las responsabilidades que ostente.